(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 February 2004 (26.02.2004)

PCT

(10) International Publication Number
WO 2004/017604 A2

(51) International Patent Classification[7]: H04L 29/06

(21) International Application Number:
PCT/US2003/025123

(22) International Filing Date: 11 August 2003 (11.08.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/222,307 15 August 2002 (15.08.2002) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 10/222,307 (CIP)
Filed on 15 August 2002 (15.08.2002)

(71) Applicant *(for all designated States except US)*: WASHINGTON UNIVERSITY IN ST. LOUIS [US/US]; One Brooking Drive, St. Louis, MO 63130 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: SCHUEHLER, David, V. [US/US]; 12306 Halsgame Lane, St. Louis, MO 63141 (US). LOCKWOOD, John, W. [US/US]; 839 Jackson Ave., St. Louis, MO 63130 (US).

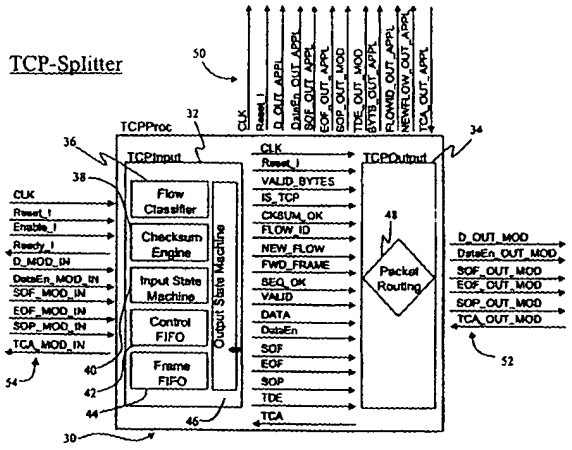
(74) Agents: RASCHE, Patrick, W. et al.; Armstrong Teasdale LLP, One Metropolitan Square, Sute 2600, St. Louis, MO 63102 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE,

[Continued on next page]

(54) Title: TCP-SPLITTER: RELIABLE PACKET MONITORING METHODS FOR HIGH SPEED NETWORKS



(57) Abstract: A method for obtaining daata while facilitating keeping a minimum amount of state is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one of an Application Specific Integrated Circuit (ASIC) and a Field Programmable Gate Array FPGA. The method also includes removing an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame with at least one of the ASIC and the FPGA, and determining a validity of a checksum of the removed steam-oriented protocol header. The method also includes dropping the IP frame when the checksum is invalid, supplying a client application with data from the removed protocol frame when the checksum is valid, and sending an IP frame including the removed stream-oriented protocol frame to the third device from the first device when the checksum is valid.

WO 2004/017604 A2

ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

(48) Date of publication of this corrected version:
3 June 2004

(15) Information about Correction:
see PCT Gazette No. 23/2004 of 3 June 2004, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

TCP-SPLITTER: RELIABLE PACKET MONITORING METHODS FOR HIGH SPEED NETWORKS

BACKGROUND OF THE INVENTION

[0001] This invention relates generally to data transfer through a network and, more particularly, to the monitoring of data passing through the Internet.

[0002] At least some known protocol analyzers and packet capturing programs have been around as long as there have been networks and protocols to monitor. These known tools provide the ability to capture and save network data with a wide range of capabilities.

[0003] For example, one such program "tcpdump" available from the Lawrence Berkeley National Laboratory (http://ee.lbl.gov/) allows for the capture and storage of TCP packets. These known tools work well for monitoring data at low bandwidth rates, but the performance of these programs is limited because they execute in software. Post processing is required with these tools in order to reconstruct TCP data streams.

[0004] Accordingly, it would be desirable to provide a solution to data monitoring that is implementable at high bandwidth rates.

BRIEF DESCRIPTION OF THE INVENTION

[0005] In one aspect, a method for obtaining data while facilitating keeping a minimum amount of state is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device. The method also includes removing an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame with the logic device, and determining a validity of a checksum of the removed steam-oriented protocol header. The method also includes dropping

the IP frame when the checksum is invalid, actively dropping IP frames such that the first device always has an accumulated in order content stream before the third device accumulates the in order stream, supplying a client application with data from the removed protocol frame when the checksum is valid, and sending an IP frame including the removed stream-oriented protocol frame to the third device from the first device when the checksum is valid.

[0006] In another aspect, an apparatus for facilitating keeping a minimum amount of state is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded stream-oriented protocol frame including a header and data packet from the received IP frame, classify the removed protocol frame as at least one of a sequence number greater than expected classification, a sequence number equal to expected, and a sequence number less than expected classification. The logic device is also configured to send an IP frame including the removed protocol frame to the second device when the classification is one of the sequence number less than expected classification and the sequence number equal to expected and drop the received IP frame including the removed protocol frame when the classification is the sequence number greater than expected classification. The logic device is also configured to actively drop IP frames such that the apparatus always has an accumulated in order content stream before the second device accumulates the in order stream.

[0007] In yet another aspect, an apparatus includes at least one input port, at least one output port, and at least one reprogrammable device. The apparatus also includes at least one logic device operationally coupled to the input port, the output port, and the reprogrammable device. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded stream-oriented protocol frame including a header and a data

packet from the received IP frame, and determine if the removed protocol frame includes programming data. The logic device is also configured to reprogram the reprogrammable device when the removed protocol frame contains programming data, and send an IP frame including the removed protocol frame to the second device.

[0008] In still another aspect, an apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame, and determine if the removed protocol frame includes data representing a quality of service (QoS) algorithm. The logic device is further configured to supply a client application with data from the removed protocol frame when the removed protocol includes QoS data, and send an IP frame including the removed protocol frame to the second device.

[0009] In yet still another aspect, a network including a plurality of switching devices operationally coupled to each other is provided. At least some of the switching devices including at least one logic device configured to monitor stream-oriented network traffic for Field Programmable Gate Array (FPGA) programming data, reprogram at least one of itself and a FPGA coupled to said logic device upon receipt of the FPGA programming data, and retransmit the FPGA programming data back onto the network such that other switching devices can reprogram themselves using the FPGA programming data.

[0010] In still yet another aspect, a method for distributing data is provided. The method including receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, wherein the first device includes an Integrated Circuit (IC). The method also includes removing an embedded protocol frame from the received IP frame with the IC, supplying a client application with data

from the removed protocol frame, analyzing the data supplied to the client application, and sending an IP frame including the removed protocol frame to the third device from the first device only after analyzing the data.

[0011] In one aspect, a method for distributing data on a network using a single TCP/IP source, a single destination and one or more intermediate hardware based monitoring nodes is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device. The method also includes removing an embedded Transmission Control Protocol (TCP) frame from the received IP frame with the logic device, supplying a client application with data from the removed protocol frame and sending an IP frame including the removed protocol frame to the third device from the first device after performing an analysis on the removed frame.

[0012] In another aspect, a method for identifying and selectively removing data on a data transmission system is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device, and actively dropping IP frames addressed to the third device sent by the second device such that the first device always has an accumulated in order content stream before the third device accumulates the in order content stream.

[0013] In yet another aspect, a dynamically reconfigureable data transmission system, having an apparatus including at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded protocol frame from the received IP frame, and determine if the removed protocol frame includes Field Programmable Gate Array (FPGA) programming data. The logic

device is also configured to supply a client application with data from the removed protocol frame when the removed protocol includes FPGA programming data such that the application receives a content stream in order, and send an IP frame including the removed protocol frame to the second device.

[0014] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a data stream, determining a particular byte offset within the monitored stream at which to block flow of the stream, and blocking flow of the data stream at the determined byte offset.

[0015] In another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a data stream for a first predetermined condition, blocking flow of the data steam upon a detection of the first predetermined condition, and re-enabling flow of the blocked stream.

[0016] In yet another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a TCP data stream for a predetermined condition, and generating and transmitting a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

[0017] In still another aspect, a method for controlling traffic on a network is provided. The method includes monitoring TCP traffic in band through a switch using a plurality of content scanning engines.

[0018] In one aspect, a method for controlling traffic on a network is provided. The method includes content scanning a plurality of TCP packets to detect a content match that spans multiple packets.

[0019] In another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of flows through the

network wherein per flow memory usage is matched to a burst width of a memory module used to monitor a flow.

[0020] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

[0021] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of data flows simultaneously, assigning a maximum idle period of time for each monitored flow, and stopping monitoring a flow which is idle for at least the assigned period of time.

[0022] In still another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of data flows simultaneously, maintaining a period of idle time for each monitored flow, and stopping monitoring the flow having a longest period of idle time.

[0023] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry, receiving a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision, and stopping monitoring of the existing flow whose hash table entry the new flow collided with.

[0024] In another aspect, A Field Programmable Gate Array (FPGA) is configured to monitor a plurality of data flows using a hash table to store state information regarding each flow, resolve hash table collisions according to a first algorithm stored on the FPGA, receive a second algorithm at the FPGA to resolve hash table collisions, the second algorithm different from the first algorithm, and use the received second algorithm to resolve hash table collisions occurring subsequent the receipt of the second algorithm.

[0025] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a data stream, determine a particular byte offset within the monitored stream at which to block flow of the stream, and block flow of the data stream at the determined byte offset.

[0026] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a data stream for a first predetermined condition, block flow of the data steam upon a detection of the first predetermined condition, and re-enable flow of the blocked stream.

[0027] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a TCP data stream for a predetermined condition, and generate and transmit a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

[0028] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a TCP data stream from a first device directed toward a second device for a predetermined condition, and manipulate the TCP data stream such that the second device receives data different than that sent from the first device.

[0029] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor TCP traffic in band using a plurality of content scanning engines.

[0030] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to scan a plurality of TCP packets to detect a content match that spans multiple packets.

[0031] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of flows through the network wherein per flow memory usage is matched to a burst width of a memory module used to monitor a flow.

[0032] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

[0033] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of data flows

simultaneously, assign a maximum idle period of time for each monitored flow, and stop monitoring a flow which is idle for at least the assigned period of time.

[0034] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of data flows simultaneously, maintain a period of idle time for each monitored flow, and stop monitoring the flow having a longest period of idle time.

[0035] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry, receive a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision, and stop monitoring of the existing flow whose hash table entry the new flow collided with.

BRIEF DESCRIPTION OF THE DRAWINGS

[0036] Figure 1 is a high level view of the data flow through an embodiment of a TCP-Splitter.

[0037] Figure 2 is a low-level view of data flow through an embodiment of a TCP-splitter.

[0038] Figure 3 is a perspective view of a Field-programmable Port Extender (FPX) module.

[0039] Figure 4 is a schematic view of the FPX module shown in Figure 3.

[0040] Figure 5 illustrates a plurality of workstations connected to a remote client across the Internet through a node including a TCP-Splitter coupled to a monitoring client application.

[0041] Figure 6 illustrates a plurality of TCP-Splitter implemented FPX modules, as shown in Figures 3 and 4, in series between a programming device and a endpoint device forming a network.

[0042] Figure 7 illustrates that monitoring systems are implemented as either in-band or out-of-band solutions.

[0043] Figure 8 illustrates a system that focuses on an in-band type of solution where network data is routed into and back out of the monitoring application.

[0044] Figure 9 illustrates a layout on an entry stored for a flow.

[0045] Figure 10 illustrates a content-scanning engine combined with the TCP-Splitter shown in Figure 1.

[0046] Figure 11 illustrates an overall throughput of the content scanner increases by putting four scanning engines in parallel and processing four flows concurrently.

[0047] Figure 12 illustrates that frames are buffered the event that there is congestion in a TCP Protocol Processing engine.

[0048] Figure 13 illustrates three packet sequencing issues.

[0049] Figure 14 illustrates an example of an overlapping retransmission and controlling signals.

DETAILED DESCRIPTION OF THE INVENTION

[0050] Figure 1 is a high level view of the data flow through an embodiment of a TCP-Splitter 10. A plurality of Internet Protocol (IP) frames 12 enter into TCP-Splitter 10 from a source device 14 and frames 12 are addressed to a destination device 16. IP frames 12 each include an IP header 18 and a TCP frame including a TCP header and a data packet. The TCP header is removed, the packet is classified to retrieve Flow State, and then the packet is sent as a byte stream to a client application 20. An IP frame including the embedded TCP frame is also sent to destination device 16. Accordingly, the splitting of the TCP frame from the IP frame is transparent to devices 14 and 16. In one embodiment, client application 20 counts how many bits are being transferred in a TCP exchange. Additionally, client application 20 is provided with TCP header information and/or IP header information, and in one embodiment, the header information is used to bill a user on a bit transferred basis. In another embodiment, client application 20 has access to reference data and, in real time, compares the byte stream of TCP transferred data provided to client application 20 with the reference data to provide a content matching such as for example but not limited content matching as described in co-pending patent application serial number 10/152,532 and now-abandoned application serial number 10/037,543, which are hereby incorporated herein in their entirety. In one embodiment, upon finding a particular match with a predefined reference data, TCP-splitter 10 stops all data flow from a particular IP address, all data flow to a particular IP address, and/or all data flow through TCP-splitter 10. In other embodiments, client application 20 monitors data through TCP-Splitter for security purposes, for keyword detection, for data protection, for copyright protection, and/or for watermark detection (and/or other types of embedded digital signatures). In one embodiment, a delay is utilized such that the data is analyzed as described above before an IP frame including the removed TCP frame is sent to the destination device. Accordingly, TCP-Splitter 10 allows for actions to be taken in real time processing. In other words, TCP-Splitter 10 allows for arbitrary actions to be taken by the client application before the IP frame

is sent to the destination device. These actions include delaying transmission of the IP frame and stopping transmission of the IP frame. Additionally, in some placements, IP frames 12 can be wrapped with other protocol wrappers such as an ATM Adaptation Layer 5 (AAL5) frame wrapper 22 and an Asynchronous transmission mode (ATM) Cell wrapper 24.

[0051] TCP-Splitter 10 is not implemented in software, rather TCP-Splitter 10 is implemented with combinational logic and finite state machines in a logic device. As used herein a logic device refers to an Application Specific IC (ASIC) and/or a Field Programmable Gate Array (FPGA), and excludes processors. In the FPGA prototype, TCP-splitter 10 processes packets at line rates exceeding 3 gigabits per second (Gbps) and is capable of monitoring 256k TCP flows simultaneously. However, TCP-Splitter 10 is not limited in the number of simultaneous TCP flows TCP-Splitter 10 is capable of monitoring and while 256k flows was implemented in the prototype built, additional flows can easily be monitored by increasing the amount of memory utilized. Additionally, TCP-splitter 10 delivers a consistent byte stream for each TCP flow to client application 20. As explained in greater detail below, TCP-splitter 10 processes data in real time, provides client application 20 the TCP packets in order, and eliminates the need for large reassembly buffers. Additionally, by providing the TCP content in order, TCP-Splitter facilitates keeping a minimal amount of state.

[0052] Figure 2 is a low level view of data flow through an embodiment of a TCP-splitter 30 illustrating a TCP input section 32 and a TCP output sections 34. Input section 32 includes a Flow classifier 36, a Checksum Engine 38, an Input State Machine 40, a Control First In-First Out (FIFO) buffer 42, a Frame FIFO buffer 44, and an Output State Machine 46. TCP output section 34, includes a Packet Routing engine 48 operationally coupled to a Client Application 50 and an IP output stack 52.

[0053] In use, data is delivered to an input stack 54 operationally coupled to TCP input section 32. Flow Classifier 36, Checksum Engine 38, Input State Machine 40, Control FIFO 42, and Frame FIFO 44 all process IP packet data received from the IP protocol wrapper. Output State Machine 46 is responsible for clocking data out of the control and frame FIFOs 42 and 44, and into output section 34. The input interface signals to TCP-Input section 32 are as follows:

IN 1 bit clock
IN 1 bit reset
IN 32 bit data word
IN 1 bit data enable
IN 1 bit start of frame
IN 1 bit end of frame
IN 1 bit start of IP payload

[0054] IP frames are clocked into input section 32 thirty-two data bits at a time. As data words are clocked in, the data is processed by Input State Machine 40 and buffered for one clock cycle. Input State Machine 40 examines the content of the data along with the control signals in order to determine the next state.

[0055] The output of Input State Machine 40 is the current state and the corresponding data and control signals for that state. This data is clocked into Flow Classifier 36, Checksum Engine 38, and Frame FIFO 44.

[0056] Flow Classifier 44 performs TCP/IP flow classification, verifies the sequence number, and maintains state information for this flow. Output signals of Flow Classifier 44 are (1) a 1 bit indication of whether or not this is a TCP packet, (2) a variable length flow identifier (currently eighteen bits), (3) a 1 bit indication of whether of not this is a new TCP flow, (4) a 1 bit indication of whether or not this packet should be forwarded, (5) a 1 bit indication of whether the sequence number was correct, and (6) a 1 bit indication of the end of a TCP flow.

[0057] Checksum Engine 38 verifies the TCP checksum located in the TCP header of the packet. The output of the checksum engine is a 1-bit indication

whether of not the checksum was successfully verified. Frame FIFO 44 stores the IP packet while Checksum Engine 38 and Flow Classifier 36 are operating on the packet. Frame FIFO 44 also stores a 1 bit indication of the presence of TCP data, a 1 bit indication of the start of frame, a 1 bit indication of the end of frame, a 1 bit indication of the start of the IP packet payload, a 1 bit indication of whether or not there is valid TCP data, and a 2 bit indication of the number of valid bytes in the data word. The packet is stored so that the checksum and flow classifier results can be delivered to the outbound section 34 along with the start of the packet.

[0058] Once the flow has been classified and the TCP checksum has been computed, information about the current frame is written to Control FIFO 42. This data includes the checksum result (pass or fail), a flow identifier (currently 18 bits), an indication of whether or not this information is the start of a new flow, an indication of whether or not the sequence number matched the expected sequence number, a signal to indicate whether or not the frame should be forwarded, and a 1 bit indication of whether or not this is the end of a flow. Control FIFO 42 facilitates holding state information of smaller frames while preceding larger frames are still being clocked out of Frame FIFO 44 for outbound processing.

[0059] Output State Machine 46 is responsible for clocking data out of the Control and Frame FIFOs 42 and 44, and into output section 34 of TCP-Splitter 30. Upon detecting a non-empty Control FIFO 42, output state machine 46 starts clocking the next IP frame out of Frame FIFO 44. This frame data along with the control signals from Control FIFO 42 exit TCP-Input section 32 and enter TCP-Output section 34.

[0060] TCP-Splitter 30 uses a flow classifier that can operate at high speed and has minimal hardware complexity. In an exemplary embodiment a flow table with a 256k element array contained in a low latency static RAM chip is used. Each entry in the table contains thirty-three bits of state information. An eighteen-bit hash of the source IP address, the destination IP address, the source TCP port, and the

destination TCP port are used as the index into the flow table. The detection of a TCP FIN flag signals the end of a TCP flow and the hash table entry for that particular flow is cleared. Other classifiers can be used to identify traffic flows for TCP-Splitter 30. For example, SWITCHGEN (as described in "Pattern Matching in Reconfigurable Logic for Packet Classification", ACM Cases, 2001, A. Johnson and K. Mackenzie) is a tool which transforms packet classification into reconfigurable hardware based circuit design and can be used with TCP-splitter 30. A Recursive Flow Classification (RFC) algorithm can also be used with TCP-Splitter and is another high performance classification technique that optimizes rules by removing redundancy. The design of TCP-Splitter 30 does not impose any restrictions on the flow classification technique utilized and can be used with any flow classifier.

[0061] In an exemplary embodiment, output-processing section 34 of TCP-Splitter 30 is responsible for determining how a packet should be processed. The input interface signals to output section 34 are as follows:

IN 1 bit clock
IN 1 bit reset
IN 32 bit data word
IN 1 bit data enable
IN 1 bit start of frame
IN 1 bit end of frame
IN 1 bit start of IP payload
IN 1 bit TCP data enable
IN 2 bit number of valid data bytes
IN 1 bit TCP protocol indication
IN 1 bit checksum passed
IN 18 bit flow identifier
IN 1 bit new flow indication
IN 1 bit forward frame indication
IN 1 bit correct sequence number
IN 1 bit data is valid
IN 1 bit end of flow

[0062] There are three possible choices for packet routing. Packets can be (1) passed on to the outbound IP stack only, (2) passed both to the outbound IP

stack and to client application 50, or (3) discarded (dropped). The rules for processing packets are as follows:

All non-TCP packets (i.e., classified as non-TCP) are sent to the outbound IP stack.
All TCP packets with invalid checksums (i.e., classified as invalid TCP checksum) are dropped.
All TCP packets with sequence numbers less than the current expected sequence number (i.e., classified as sequence number less than expected) are sent to the outbound IP stack.
All TCP packets with sequence numbers greater than the current expected sequence number (i.e., classified as sequence number greater than expected) are dropped (i.e., discarded and not sent to either client application 50 or the outbound IP stack).
All TCP synchronization (TCP-SYN) packets are sent to the outbound IP stack.
All other packets (classified as else) are forwarded both to the outbound IP stack and client application 50. Note that when the TCP packet has a sequence number equal to expected and has a valid checksum, then that packet is classified as else and sent to the outbound IP stack as well as to client application 50.

[0063] A client interface (not shown) is between client application 50 and TCP output section 34. The client interface provides a hardware interface for application circuits. Only data that is valid, checksummed, and in-sequence for each specific flow is passed to client application 50. This allows the client to solely process the consistent stream of bytes from the TCP connection. All of the packet's protocol headers are clocked into client application 50 along with a start-of-header signal so that the client can extract information from these headers. This eliminates the need to store header information, but still allows the client access to this data. Client application 50 does not sit in the network data path and therefore does not induce any delay into the packets traversing the network switch. This allows the client application to have arbitrary complexity without affecting the throughput rate of TCP-splitter 30. The client interface contains the following signals:

IN 1 bit clock
IN 1 bit reset
IN 32 bit data word
IN 1 bit data enable
IN 1 bit start of frame
IN 1 bit end of frame

IN 1 bit start of IP payload
IN 1 bit TCP data enable
IN 2 bit number of valid data bytes
IN 18 bit flow identifier
IN 1 bit new flow indication
IN 1 bit end of flow
OUT 1 bit flow control

[0064] Client application 50 can generate a flow control signal that will stop the delivery of cells. In one embodiment, this signal is not processed by TCP-Splitter 30, but is passed on to the IP wrapper driving the ingress of IP packets. In another embodiment, this signal is processed by TCP-Splitter 30.

[0065] In the embodiment where TCP-Splitter 30 does not process the flow control signals, there is a delay in the cessation of the flow of data words into client application 50 while the flow control signal is being processed by the lower protocol layers. Since TCP-Splitter 30 does not act upon the flow control signal, data continues to flow until all buffers of TCP-Splitter 30 are empty. Client application 50 is configured to either handle data at line rates or is capable of buffering 1500 bytes worth of data after the flow control signal is asserted.

[0066] Because Transmission Control Protocol / Internet Protocol (TCP/IP) is the most commonly used protocol on the Internet, it is utilized by nearly all applications that require reliable data communications on a network. These applications include Web browsers, FTP, Telnet, Secure Shell, and many other applications. High-speed network switches currently operate at OC-48 (2.5Gb/s) line rates, while faster OC-192 (10 Gb/s) and OC-768 (40 Gb/s) networks will likely be implemented in the near future. New types of networking equipment require the ability to monitor and interrogate the data contained in packets flowing through this equipment. TCP-Splitter 30 provides an easily implementable solution for monitoring at these increased bandwidths.

[0067] In one embodiment, and as explained in greater detail below, TCP-Splitter 30 provides a reconfigurable hardware solution which provides for the

monitoring of TCP/IP flows in high speed active networking equipment. The reconfigurable hardware solution is implemented using Very High Speed Integrated Circuit (VHSIC) Hard-ware Description Language (VHDL) for use in ASICs or Field Programmable Gate Arrays (FPGAs). The collection of VHDL code that implements this TCP/IP monitoring function is also called TCP-Splitter in addition to the hardware itself (i.e., 10 and 30). This name stems from the concept that the TCP flow is being split into two directions. One copy of each network data packet is forwarded on toward the destination host. Another copy is passed to a client application monitoring TCP/IP flows. In an alternative embodiment, the copy that is passed to the client application is rewrapped with an IP wrapper to form an IP frame that is forwarded to the destination host. In order to provide for the reliable delivery of a stream of data into a client application, a TCP connection only needs to be established that transits through the device monitoring the data. The bulk of the work for guaranteed delivery is managed by the TCP endpoints, not by the logic on the network hardware. This eliminates the need for a complex protocol stack within the reconfigurable hardware because the retransmission logic remains at the connection endpoints, not in the active network switch.

[0068] TCP-Splitter 30 is a lightweight, high performance circuit that contains a simple client interface that can monitor a nearly unlimited number of TCP/IP flows simultaneously. The need for reassembly buffers is eliminated because all frames for a particular flow transit the networking equipment in order. Because there is no guarantee that TCP frames will traverse the network in order, some action will have to take place when packets are out of order. As explained above, by actively dropping out of order packets, a TCP byte stream is generated for the client application without requiring reassembly buffers. If a missing packet is detected, subsequent packets are actively dropped until the missing packet is retransmitted. This ensures in-order packet flow through the switch. Therefore a monitoring device always has an accumulated in order content stream before a destination device accumulates the in order content stream.

[0069] This feature forces the TCP connections into a Go-Back-N sliding window mode when a packet is dropped upstream of the monitoring node (e.g., the node where TCP-Splitter is positioned). The Go-Back-N retransmission policy is widely used on machines throughout the Internet. Many implementations of TCP, including that of Windows 98, FreeBSD 4.1, and Linux 2.4, use the Go-Back- N retransmission logic. The benefit on the throughput is dependent on the specific TCP implementations being utilized at the endpoints. In instances where the receiving TCP stack is performing Go-Back-N sliding window behavior, the active dropping of frames may improve overall network throughput by eliminating packets that will be discarded by the receiver.

[0070] Typically, TCP-Splitter 30 is placed in the network where all packets of monitored flows will pass. All packets associated with a TCP/IP connection being monitored then passes through the networking node where monitoring is taking place. It would otherwise be impossible to provide a client application with a consistent TCP byte stream from a connection if the switch performing the monitoring only processed a fraction of the TCP packets. In general, this requirement is true at the edge routers but not true for interior nodes of the Internet. This strategic placement of TCP-Splitter can be easily accomplished in private networks where the network has been designed to pass traffic in a certain manner.

[0071] Figure 3 is a perspective view and Figure 4 is a schematic view of a Field-programmable Port Extender (FPX) module 60 configured to implement a TCP-Splitter as described above. Module 60 includes a Network Interface Device (NID) 62 operationally coupled to a Reprogrammable Application Device (RAD) 64. NID 62 is configured to program and/or reprogram RAD 64. Module 60 also includes a plurality of memories including a static RAM 66, a Synchronous DRAM 68, and a PROM 70 operationally coupled to at least one of NID 62 and RAD 64. In an exemplary embodiment, NID 62 includes a FPGA such as a

XCV600E FPGA commercially available from Xilinx, San Jose CA, and RAD 64 includes a FPGA such as a XCV2000E FPGA also available from Xilinx.

[0072] In use, module 60 monitors network traffic and send TCP data streams to a client application in order. Because module 60 implements the TCP-Splitter in a FPGA, upon receipt of FPGA programming data the TCP-Splitter can reprogram itself and send the FPGA programming data to other TCP-splitters in a flow path between a sending device and a destination device. Additionally, module 60 can reprogram the router to process the traffic flow differently than before. Also, because the data is passed along to other TCP-Splitters, the overall QoS of a network is quickly and easily changeable. In other words, QoS policies, as known in the art, are easily and quickly changed in networks including a TCP-Splitter as herein described. Accordingly, the reprogrammed router prioritizes network traffic based on the flow. Additionally, the TCP-Splitter can include a plurality of reprogrammable circuits such as FPGAs and can monitor the TCP flows for different things substantially simultaneously. For example, one flow contains data in order for a client application to count bits, while another flow contains data in order for another client application to perform content matching, while another flow contains data for reprogramming an FPGA. Also, a plurality of FPGAs can be coupled to each other such that upon receipt by at least one of an ASIC and a FPGA of FPGA programming data, the ASIC or FPGA receiving the data uses the data to reprogram the FPGAs.

[0073] Figure 5 illustrates a plurality of workstations 80 connected to a remote client 82 across the Internet 84 through a node 86 including a TCP-Splitter (not shown in Figure 4) coupled to a Monitoring client application 88. All traffic from remote client 82 or any other device on the Internet 84 to or from workstations 80 passes through node 86 and is monitored with the TCP-Splitter coupled to client application 88.

[0074] Figure 6 illustrates a plurality of TCP-Splitter implemented FPX modules 60 (Shown in Figures 3 and 4) in series between a programming device 90 and an endpoint device 92 forming a network 94.

[0075] In use, programming device 90 transmits programming data via a stream-oriented protocol using the Internet Protocol (IP) to send the data to endpoint device 92. Each FPX module 60 receives a plurality of IP frames addressed to endpoint device 92, removes the embedded stream-oriented protocol frame from the IP frame, and provides a client application the removed stream-oriented protocol frame. Each FPX module 60 sends an IP frame including the removed protocol frame back onto network 92. Accordingly, with one transmission stream made from programming device 90 to endpoint device 92, a plurality of intermediate devices (modules 60) receive programming data either to reprogram themselves (modules 60) or to reprogram any attached devices. Because the programming data is split (i.e., sent to the client application and sent back on network 94 addressed to endpoint device 92), TCP-Splitter imparts the ability to easily and quickly reconfigure a network of any size. As used herein, a stream-oriented protocol refers to all protocols that send data as a stream of packets such as TCP as opposed to non-stream-oriented protocols such as UDP where a single packet contains the entire message.

[0076] The above described TCP-Splitter is a circuit design which supports the monitoring of TCP data streams. A consistent byte stream of data is delivered to a client application for every TCP data flow that passes through the circuit. The TCP-Splitter accomplishes this task by tracking the TCP sequence number along with the current flow state. Selected out-of-order packets are dropped in order to provide the client application with the full TCP data stream without requiring large stream reassembly buffers. The dropping of packets to maintain an ordered flow of packets through the network has the potential to adversely affect the overall throughput of the network. However, an analysis of out-of-sequence packets in Tier-1 IP backbones has found that approximately 95% of all TCP packets were

detected in proper sequence. Network induced packet reordering accounted for a small fraction of out-of-sequence packets, with the majority resulting from retransmissions due to data loss. Greater than 86% of all TCP flows observed contained no out-of- sequence packets.

[0077] The first implementation of the above described TCP-Splitter stored 33 bits of state information for each active flow in the network. By utilizing a low latency SRAM module, 256k simultaneous flows were supported. The TCP-Splitter circuit utilized 32 bit wide data path of the FPX card and could operate at 100MHz. At that clock rate, a maximum throughput of 3.2Gbps was supported.

[0078] Monitoring systems are implemented as either in-band or out-of-band solutions. The two basic types are illustrated in Figure 7. Out-of-band solutions are always passive systems. For example, a cable splitter, network tap, or other content duplicating mechanism is employed to deliver a copy of the network traffic to monitoring system. Since the monitor processes a copy of the true network traffic, it has no means by which to alter the network content and is therefore always a passive solution. In contrast, in-band network monitoring systems are positioned within the data path of the network. Traffic is processed and forwarded on toward end systems. In-band solutions could be configured to alter the content of the network by either inserting, dropping, or modifying network traffic. The following system is developed as an in-band monitoring device in order to provide a flexible platform upon with either passive or active extensible networking solutions can be developed.

[0079] The above described TCP-Splitter design was more closely aligned to the out-of-band type of monitoring solution. Network data was duplicated and a copy was passed to the monitoring application. Figure 8 illustrates a system 100 that focuses on an in-band type of solution where network data is routed into and back out of the monitoring application. Developing a monitoring solution in this manner allows one to block selected flows, unblock previously blocked flows, and alter flow specific data as the data traverses through the monitoring system 100. Blocking flows

allows system 100 to prevent data (ie. a virus) from reaching its intended destination. The unblocking of flows allows one to block the transmission of data (ie. confidential or copyrighted material) until an authorizing authority has granted permission for the blocked content to be delivered to the end user selectively. The altering of data is useful in removing a virus that has attached itself to normal (proper) data transiting the network. In this manner, a virus attached to a web page or email can be removed from the network without preventing the user from receiving the desired content. An example of an authorizing authority includes a Regional Transaction Processor (RTP) (not shown) as described in co-pending application serial No. 10/037,593. The RTP includes various hardware and software components (not shown) that allow the RTP to communicate with system 100, facilitate and process transactions, and store the related information securely over the Internet. A remote access server (not shown) can receive content match information directly from content owners via communication lines connected to the content owners' servers (not shown). A DB server (not shown) and storage system (not shown) store the content match information. A Central Storage and Backup System (CSBS) (not shown) backs up and stores data from one of more RTPs. The CSBS receives data from the RTPs through a router (not shown), a firewall (not shown), and an enterprise switch (not shown) to back-up onto a plurality of storage systems (not shown). A DataBase (DB) server (not shown) date stamps and logs all information received.

[0080] The RTP that processes transactions based on messages from a workstation (not shown) and/or system 100. An accounting server (not shown) receives and processes transactions using data in the DB server, a remote access server (not shown), and an external billing system (not shown) to facilitate transactions at the workstation.

[0081] The RTP has access to a tax lookup table (not shown) stored on the DB server or the storage system. The tax table can be used to determine the amount of sales tax rates to add to the price of delivering content through the system

100 or other retail transactions made by a user at the workstation. An identifier for the workstation and an identifier for the corresponding system 100 can be used to determine which tax tables or tax rate formula to use to determine the amount of state and/or local sales tax to charge for a transaction.

[0082] System 100 and the RTP also can use public/private key encryption/decryption technology to decrypt encrypted data packets for content matching, and then re-encrypt the data packet and forward it to the workstation. Alternatively, content match information can be provided to identify the content of a data packet in encrypted format, thereby eliminating the need to decrypt the data packet.

[0083] The content provider also can supply or indicate transaction instructions to be used in the RTP when the system 100 finds a content match in a data packet. For example, if the user is required to pay for the content before receiving it, the RTP transmits a transaction prompt to the user's workstation (not shown) informing the user of the price to be paid for the content, and allowing the user to accept or decline the purchase. As another example, the RTP can transmit a prompt to inform the user that content infected with a virus is attempting to be transmitted from or received to the user's workstation, and that transmission or reception of the virus is being halted. As another example, the RTP can transmit a prompt to inform the user that content subject to security control is attempting to be transmitted from or received to the user's workstation, and that transmission or reception of the confidential content is being halted. As a further example, the RTP can tally statistics regarding transmission of designated content for purposes such as rating the popularity of the content.

[0084] System 100 performs content match searches for content unique to application(s) being performed by various entities such as content providers, business organizations, and/or government organizations. The applications can handle various situations based on the content, such as collecting payment for the

authorized use of copyrighted content, preventing outgoing transmission of confidential material, preventing incoming receipt of material previously designated as unwanted by the recipient, and preventing the transmission of files containing viruses.

[0085] The content matches can include logic to search for particular watermarks or fingerprints to identify copyright content based on content match information from the content providers. The content matches also can include logic to understand one or more hashing functions.

[0086] System 100 includes a hardware circuit which is capable of reassembling TCP/IP data flows into their respective byte streams at multi-gigabit line rates. A large per-flow state store maintains 64 bytes of state information for millions of active TCP flows concurrently. Additional logic provides support for flow blocking, unblocking and stream modification features. System 100 enables a new generation of network services to operate within the core of the Internet.

[0087] System 100 includes a plurality of content scanner engines 102. Each content-scanning engine 102 is a hardware module that is capable of scanning the payload of packets for a set of regular expressions as described by J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos in *Implementation of a Content-Scanning Module for an Internet Firewall* IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, April 2003 and co-pending U.S. Patent application serial No. 10/152,532.

[0088] Regular expressions are well-known tools for defining conditional strings. A regular expression may match several different strings. By incorporating various regular expression operators in a pattern definition, such a pattern definition may encompass a plurality of different strings. For example, the regular expression operator ".*" means "any number of any characters". Thus, the regular expression "c.*t" defines a data pattern that encompasses strings such as "cat",

"coat", "chevrolet", and "cold is the opposite of hot". Another example of a regular expression operator is "*" which means "zero or more of the preceding expression". Thus, the regular expression "a*b" defines a data pattern that encompasses strings such as "ab", "aab", and "aaab", but not "acb" or "aacb". Further, the regular expression "(ab)*c" encompasses strings such as "abc", "ababc", "abababc", but not "abac" or "abdc". Further still, regular expression operators can be combined for additional flexibility in defining patterns. For example, the regular expression "(ab)*c.*z" would encompass strings such as the alphabet "abcdefghij klmnopqrstuvwxy", "ababcz", "ababcqsrz", and "abcz", but not "abacz", "ababc" or "ababacxvhgfjz".

[0089] As regular expressions are well-known in the art, it is unnecessary to list all possible regular expression operators (for example, there is also an OR operator "|" which for "(a|b)" means any string having "a" or "b") and combinations of regular expression operators. What is to be understood from the background material described above is that regular expressions provide a powerful tool for defining a data pattern that encompasses strings of interest to a user of system 100.

[0090] To accomplish the scanning of the payload of packets for a set of regular expressions, each content-scanning engine 102 employs a set of Deterministic Finite Automata (DFAs), each searching in parallel for one of the targeted regular expressions. Upon matching the content of a network data packet with any of these regular expressions, content-scanner 102 has the ability to either allow the data to pass or to drop the packet. Content-scanning engine 102 also has the ability to generate an alert message that can be sent to a log server when a match is detected in a packet. In an exemplary embodiment, the alert message contains the source and destination addresses of the matching packet along with a list of regular expressions that were found in the packet.

[0091] The TCP based content scanning engine integrates and extends the capabilities of the above described TCP-Splitter and Content-Scanning Engine 102. As illustrated in Figure 8, data flow is from the left to the right. IP packets are passed into a TCP Protocol Processing Engine 104 from lower layer protocol processing engines contained within a network switch. An input packet buffer 106 provides an amount of packet buffering when there are downstream processing delays. TCP Protocol Processing Engine 104 validates packets and classifies them as part of a flow. Packets along with the associated flow state information are passed onto a Packet Routing module where the packets are routed either to Content Scanning Engines 102 or a Flow Blocking module 108. Multiple Con-tent Scanning Engines 102 evaluate regular expressions against the TCP data stream. Packets returning from Content Scanning Engines 102 are passed to Flow Blocking module 108 where application specific state information is stored and flow blocking is enforced. Packets that are not blocked are passed back to the network switch which forwards them on toward their end destination.

[0092] To enable a quick access for storing and retrieving state information, a hash table is used in one embodiment. However, whenever a hash table is used, hash table collisions can occur.

[0093] Gracefully handling hash table collisions is a difficult problem for real-time network systems. An efficient method for dealing with hash collisions is to have the new flow age out the previous flow whenever a collision occurs. In other words when a new flow hashes to the same value as a previous flow, the monitoring of the previous flow is stopped and the new flow is monitored. This type of action leads to the incomplete scanning of TCP flows because the context scanning engine will loose the context information of the previous flow when it encounters a new flow with the same flow identifier. To ensure all flows are properly monitored, a linked list of flow state records can be chained off of the appropriate hash entry. The advantage to this approach is that all flows that encounter hash

collisions in the state store can be fully monitored. The major drawback to this approach is that the time required to traverse a linked list of hash bucket entries could be excessive. The delay caused in retrieving flow state information can adversely affect the throughput of the system and lead to data loss. Another drawback of linked entries in the state store is the need to perform buffer management operations. This induces additional processing overhead into a system which is operating in a time critical environment. A State Store Manager 110 overcomes this problem by limiting the length of the chain to a constant number of entries.

[0094] A hashing algorithm which produces an even distribution across all hash buckets is important to the overall efficiency of the circuit. Initial analysis of the flow classification hashing algorithm used for system 100 was performed against packet traces available from the National Laboratory for Applied Network Research. With 26,452 flow identifiers hashed into a table with 8 million entries, there was a hash collision in less than .3% of the flows.

[0095] Additional features of system 100 includes support for the following services: Flow Blocking which allows a flow to be blocked at a particular byte offset within the TCP data stream); Flow Unblocking in which a previously disabled flow can be re-enabled and data for a particular flow will once again be allowed to pass through the circuit; Flow Termination in which a selected flow will be shut down by generating a TCP FIN packet; and Flow Modification which provides the ability to sanitize selected data contained within a TCP stream. For example, a virus is detected and removed from a data stream. The concept of altering the content of a TCP data stream is counterintuitive to most (if not all) networks. In the majority of circumstances, this type of behavior should be avoided at all costs. But there are a selected number of situations where modifying a TCP flow can be advantageous. One such situation is the above described removal of a virus from a TCP data stream in order to prevent the spreading of the virus. Another use could be associated with a series of extensible network solutions where information is added to or removed from

a TCP data flow as the data traverses network nodes running extensible networking solutions.

[0096] There are three separate situations that need to be addressed when altering TCP stream data within the core of the network. The first involves modifying data within an existing data flow. In this case, the total number of data bytes transmitted by the source will be identical to the total number of data bytes received at the destination, only the content will have changed. The second case involves the addition of data to the data stream. In this case, the total number of data bytes received by the destination will be greater than the number of data bytes sent by the source. The third case involves the removal of data form the data stream. Here, the total number of data bytes received at the destination will be less than the total number of data bytes send by the source.

[0097] Case 1: Modifying the flow - In this situation, the processing engine which is altering the content of the TCP data stream need only operate on the flow of data in a single direction, from source to destination. When it is determined that data bytes should be altered, existing data bytes are replaced with new bytes. The TCP checksum of the network packet containing the altered data is recomputed to account for the new data. In addition, the processing engine remembers (1) the new content and (2) the TCP sequence number pertaining to the location in the data stream where the new content replaces existing content. This step is desired to handle the case where a retransmission occurs which contains data that has been altered. In order to ensure that the end system receives a consistent view of the new data stream, the old data needs to be replaced with new data whenever the old data transits the network. In this manner, the end system will always receive a consistent view of the transmitted byte stream with the selected data alterations applied.

[0098] Case 2: Adding data to the flow - When a processing engine within the network wishes to add content to a TCP data stream, the processing engine must process TCP packets sent in the forward direction from source to destination and

TCP packets sent in the reverse direction, from destination back to the source. Without processing both directions of the data flow, the system will be unable to accurately manage the insertion of data into the flow. Once the position within the network stream where the data should be inserted is realized, the processing engine can then either modify existing TCP data packets and/or generate additional TCP data packets as necessary to insert data into the data stream. For each of these packets, the appropriate TCP header fields will have to be populated, including a checksum value. Sequence numbers contained in TCP packets received by the processing engine that occur after the point of insertion within the TCP data stream are incremented by the total number of bytes that were inserted into the stream. The processing engine stores the sequence number of the location where the data insertion took place along with the total number of bytes inserted into the stream. If a packet retransmission occurs, the processing engine performs the steps taken to insert the additional stream data so that the receiving node always receives a consistent view of the amended data stream. When processing TCP packets sent back from the receiving host, the processing engine decrements the acknowledgment number whenever the acknowledgment number exceeds the sequence number where the data insertion has taken place. In this manner, the processing engine can ensure that the source node will not receive acknowledgments for data that the receiving system has not yet processed. In addition, since the processing engine is inserting new data content into the stream, the processing engine also tracks the TCP processing state of the end systems and generates retransmission packets for the inserted data whenever it detects a non-increasing sequence of acknowledgement numbers in the range of the inserted data.

[0099] Case 3: Removing data from the flow - When a processing engine within the network wishes to remove content from a TCP data stream, the processing engine processes TCP packets sent in the forward direction from the source to the destination and TCP packets sent in the reverse direction, from the destination back to the source. Without processing both packets traveling in both directions of the data flow, the system will be unable to accurately manage the

removal of data from the flow. Once the position within the TCP data stream where data should be removed is encountered, the processing engine can start the removal process by eliminating packets or shrinking the overall size of a packet by removing part of the data contained within the packet. Packets which are modified must have their length fields and checksum values recomputed. Sequence numbers contained in TCP packets received by the processing engine that occur after the point of data removal are decremented by the total number of bytes that were removed from the stream. The processing engine stores the sequence number of the location where the data removal took place along with the total number of bytes that were removed from the stream. If a packets retransmission occurs, the processing engine performs the steps previously taken to effect the removal of data from the stream so that the receiving node always receives a consistent view of the altered data stream. When processing TCP packets sent from the receiving host back to the sending host, the processing engine increments the acknowledgment number whenever the acknowledgment number exceeds the sequence number where the data removal has taken place. In this manner, the processing engine can ensure that the source node receives the proper acknowledgment for all of the data received by the end system. Failure to perform this step could cause excessive retransmissions or a blocking of the flow of data if the amount of data removed exceeds the window size in use by the source node.

[00100] At any given moment, a high speed router may be forwarding millions of individual traffic flows. To support this large number of flows along with a reasonable amount of state information stored for each flow, a 512MB Synchronous Dynamic Random Access Memory (SDRAM) module can be utilized. The memory interface to this memory module has a 64 bit wide data path and supports a maximum burst length of eight operations. By matching system 100's per flow memory usage to the burst width of the memory module, one can optimize the memory bandwidth. Storing 64 bytes of state information for each flow optimizes the use of the memory interface by matching the amount of per flow state information

with the amount of data in a burst transfer to memory. This configuration provides support for eight million simultaneous flows. Assuming $100.00 as a purchase price for a 512MB SDRAM memory module, the cost to store context for eight million flows is only 0.00125 cents per flow or 800 flows per penny as of August 2003. Memory modules other than SDRAM are also employable.

[00101] Of the 64 bytes of data stored for each flow, TCP processing engine 102 utilizes 32 bytes to maintain flow state and memory management overhead. The additional 32 bytes of state store for each flow holds the application data for each flow context. The layout of a single entry is illustrated in Figure 9. A portion of this per-flow state storage is used to maintain the TCP data stream re-assembly operation similar to the above described TCP-Splitter design. Another portion of the storage area is used to ensure that network data packets are associated with the proper flow stored in the state store. Yet another portion of memory is used to maintain navigation information and memory management overhead. The final portion of the per-flow state storage area is used to store per-flow context information used by the monitoring application. Additional features support passing this saved context information to the monitoring application for each network data packet. Updates to a flow's context information by the monitoring application is written back to the state store so that this information can be provided back to the monitoring application when future packets for the flow are encountered. For example, the source and destination IP addresses along with the source and destination TCP ports could be hashed into a 23 bit value. This hash value could then be used as a direct index to the first entry in a hash bucket. The hash table would then contain 4 million records at fixed locations and an additional 4 million records that could be used to form a linked list. The IP addresses and ports could be hashed to a bit value other than 23 bits. For example, the IP addresses and ports could be hashed to any bit value based upon a desired number of flows to be indexed. The use of linked list records will enable the storing state information for multiple flows that hash to the same bucket. To ensure that system 100 is able to maintain real-time behavior, the number of link traversals is

constrained by a constant. And as used herein, the term "hash table collision" refers to the situation where two flows hash to the same value, and to the situation where after two flows hash to the same hash value using a hash table employing a chaining of predetermined length and a chain is full for that hash value.

[00102] State Store Manager 110 can cache state information utilizing on-chip block RAM memory. This provides faster access to state information for the most recently accessed flows. A write-back cache design provides for improved performance.

[00103] If a match spans across multiple packets, the original design of the content-scanning engine would fail to detect the match. To alleviate this problem, the new content-scanning engine processes streams from the TCP Processing Engine.

[00104] The use of the TCP Processing Engine also requires that the content scanner process interleaved flows. Because each content scanner only holds the state of one flow, it needs to be able to save and restore the current state of a flow and perform a context switch whenever a new flow arrives. When a packet arrives at the content scanner on some flow, the content scanner must restore the last known matching state for that flow. When the content scanner has finished processing the packet, it must then save the new matching state of the flow which can be done by using the state store resources of the TCP processing circuit.

[00105] Figure 10 shows the design of the content-scanning engine combined with the TCP-Splitter. When a new packet arrives from the TCP Processing Engine, it arrives concurrently with a flow ID and state information for that context. Once the state of the search engine is loaded, the content scanner can process the packet. If no matches are found in the packet, then the packet is allowed to pass through the module. If a match is discovered in the packet, then the packet may be dropped or the whole flow may be blocked. The content scanner also has the

ability to send out an alert message if a match occurs. The format of the alert message is a UDP datagram which is also used for logging events in the system. As shown in Figure 11, the overall throughput of the content scanner increases by putting four scanning engines in parallel and processing four flows concurrently. Incoming packets are dispatched to one of the scanning engines based on the last two bits of a flow ID provided by the TCP Processing Engine. By dispatching packets in this fashion, the possibility of hazards that may occur when two scanners are processing packets from the same flow simultaneously can be eliminated.

[00106] As was shown in Figure 8, data is received on the left from the Internet Protocol Wrappers and passed into input buffer 106. Frames are buffered here in the event that there is congestion in a TCP Protocol Processing engine 150 shown in Figure 12. This congestion can occur at instants when there are hash table collisions and the State Store Manager has to walk through a linked list in order to locate the proper flow context.

[00107] From input buffer 106, IP frames are passed to TCP Protocol Processing Engine 150. An input state machine 152 tracks the processing state within a single packet. Data is forwarded to (1) a Frame FIFO 154 which stores the packet, (2) a checksum engine 156 which validates the TCP checksum, and (3) a flow classifier 158. Once flow classifier 158 has computed a hash value for the packet, information is passed to State Store Manager 110 which retrieves the state information associated with the particular flow. Results are written to a Control FIFO 162 and the state store is updated with the current state of the flow.

[00108] An Output State Machine 164 reads data from the Frame and Control FIFOs and passes it to a packet routing engine 166 (shown in Figure 8). Most traffic flows through the Content Scanning Engines 102 where the data is scanned. Packet retransmissions bypass Content Scanning Engines 102 and are sent directly to Flow Blocking module 108.

[00109] Data returning from Content Scanning Engines 102 is passed to Flow Blocking module 108. At this stage, the per flow state store is updated with the latest application specific state information. If flow blocking is enabled for a flow, it is enforced at this time. The sequence number of the packet is compared with the sequence number where flow blocking should take place. If the packet meets the blocking criteria, it is dropped from the network at this point. Packets that are not dropped are passed on to the outbound Protocol Wrapper.

[00110] State Store Manager 110 is responsible for processing requests for and updates to a flow state record. All interactions with a SDRAM memory 166 are handled along with the caching of recently accessed flow state information. A SDRAM controller 168 exposes three memory access interfaces, a read-write interface, a write only interface, and a read only interface. Requests to these interfaces are prioritized in the same order, with the read-write interface having the highest priority.

[00111] The layout of State Store Manager 110 along with its interactions to memory controller 168 and other modules in the TCP Processing Engine are illustrated in Figure 12. Upon processing a new packet, a flow identifier hash value is computed and a record retrieval operation is initiated. State Store Manager 110 utilizes the read interface of memory controller 168 to retrieve the current state information for the flow and returns this information to the protocol processing engine. If the packet is determined to be valid and is accepted by the engine, an update operation is performed to store the new flow state. The flow blocking module also performs a SDRAM read operation in order to determine the current flow blocking state. If the flow blocking state has changed or there is an update to the application specific state information, a write operation is also performed to date the flow's saved state information.

[00112] In a worse case scenario, where there is at most a single entry per hash bucket, a total of two read and two write operations to SDRAM are

required for each packet. These operations are an eight word read to retrieve flow state, an 8 word write to initialize a new flow record, a 4 word read to retrieve flow blocking information, and a 5 word write to update application specific flow state and blocking information. No memory accesses are required for TCP acknowledgment packets that contain no data. Analysis indicates that all of the read and write operations can be performed during the packet processing time if the average TCP packet contains more than 120 bytes of data. If the TCP packets contain less than this amount of data, insufficient time may be available to complete all of the memory operations while processing the packet. If this occurs, the packet may be stalled while waiting for a memory operation to complete. The average TCP packet size on the Internet has been shown to be approximately 300 bytes. It is important to note that the TCP Protocol Processing engine does not need to access memory for acknowledgment packets that contain no data. Given that half of all TCP packets are acknowledgments, the average size of a packet requiring memory operations to the state store will be larger than the 300 byte average previously stated. Processing larger packets decrease the likelihood of throttling due to memory access latency. On average, the system will have over twice the memory bandwidth required to process a packet when operating at OC-48 rates.

[00113] This paper discusses architecture for performing content scanning of TCP flows within high-speed networks. The circuit design is targeted for the Xilinx XCV2000E FPGA in the FPX platform with an operational clock frequency of 80MHz. This provides for the monitoring of eight million simultaneous TCP flows at OC-48 (2.5 Gb/s) line rates. Utilizing a 512MB commodity SDRAM memory, 8M flows can be stored with at a cost of 0.00125 cents per flow as of August 2003. By storing 64 bytes per flow, it is possible to maintain the context of the scanning engine for each flow.

[00114] By developing a circuit that operates in a Field Programmable Gate Array (FPGA) device, run-time changes can be made to the list of

scanned content. Having the ability to quickly react to new filtering requirements, makes this architecture an ideal framework for a network based Intrusion Detection System.

[00115] New FPGA devices are available which have 4 times the number of logic gates and operate at over twice the clock rate of the XVC2000E used on the FPX platform. The latest memory modules support larger densities, higher clock frequencies, and Double Data Rate (DDR) transfer speeds. Utilizing these new devices, the TCP based content scanning engine could achieve OC- 192 (10 Gb/s) data rates without requiring major modifications.

[00116] The goal of a TCP based flow monitoring system is to produce a byte stream within the interior of the network which is identical to the byte stream processed by the end system. In order to do this, one must effectively track the TCP processing state of the end system and perform similar operations. The difficulty of this task stems from the fact that the traffic observed at the monitoring node could be quite different from the traffic received at the end system. Three potential packet sequencing issues are shown in Figure 13 and outlined below. 1) Packets processed at the monitoring station are not processed by the end host system (A in Figure 13). This can occur when a packet is dropped between the monitoring station and the end system. Once arriving at the monitoring system, the packet is processed accordingly and the processing state is advanced under the assumption that the end system will follow the same behavior when it received the packet. If the packet never arrives at the end system, then the state of the monitor and end system are inconsistent with respect to each other. 2) Packets processed at the end host system are not processed by the monitoring station (B in Figure 13). This can occur when successive packets of a data flow take different paths through the network. If the monitoring station is placed at a point where it sees packets traversing one path but not the other, then it will be difficult to impossible to track the state of the end system depending on what data is sent over which path. And 3) Packets processed at the monitoring station in

the order [1][2][3] may arrive at arrive at the end system in a different order (C in Figure 13). Without knowing the specifics of the protocol implementation, the monitoring system will be unable to determine how the end system processes that sequence. Even worse, the monitoring system will have no idea that the packets have been processed by the end system in a different order.

[00117] The task of maintaining per-flow state information is difficult when the following three constraints are imposed: (1) provide storage for tens of bytes of per-flow state information, (2) support millions of simultaneous flows, (3) operate within a high-speed networking environment. Eliminating any one of these constraints greatly simplifies the problem. Reducing the amount of state information required for each flow down to one bit or reducing the number of flows to less then about 100,000 allows the use of a commodity static RAM devices-or on chip memories. Eliminating the high speed networking environment would allow for long delays associated with slower memory or secondary storage. In a worse case scenario containing a steady stream of 64 byte packets, the monitoring system will only have 200ns in which to perform the required memory operations when processing data on an OC-48 link (2.5Gbps). Each packet will require a read and a write operation in order to retrieve flow context information and to store the updated flow state.

[00118] When tracking large numbers of network flows, it is impossible to utilize a directly indexed state store. A unique flow is determined by a 32 bit source IP address, a 32 bit destination IP address, a 16 bit source TCP port number, and a 16 bit destination TCP port number. This would require 2^{96} or $8*10^{28}$ individually addressable memory locations. In order to reduce this to a reasonable number, a hashing scheme or other reduced memory indexing scheme is implemented. If the hash table is sparsely populated and there are no hash collisions, then state information can be accessed in a timely manner. When dealing with large numbers of flows (on the order of a million), the issue of hash collision becomes a real concern. Take for instance, the case where there 100 different flows that all hash to the same

entry. The state retrieval algorithm will have to traverse 50 entries on average in order to navigate to the correct entry. This can lead to an excessive amount of time spent retrieving state information which leads to diminished throughput of the system. This can quickly lead to data loss and dropped packets when buffering resources are exhausted. Artificially limiting the number of entries posted to any single hash bucket can greatly improve the worst case performance associated with a lookup operation. The resulting negative tradeoff is that flow state information may be discarded for an active flow when a hash bucket reaches the limit. Content scanning and flow re-assembly operations will be interrupted when this occurs.

[00119] Additionally, TCP based network flows do not always produce a proper termination sequence. This improper termination can be caused by a system crash, power outage, a network event, or something as simple a disconnected cable. Because TCP connections can exist for long periods of time without the presence of network traffic, it is difficult for a monitoring station to determine whether a flow is idle, or if the flow should be terminated. Not terminating flows leads to the exhaustion of flow tracking resources. Prematurely terminating an active flow can lead to situations where data is allowed to traverse the network unmonitored. The problem is even worse when attempting to monitor a series of individual UDP data packets as a data stream. The UDP protocol does not contain any provisions for marking the start or end of a flow.

[00120] Due to the tight timing constrains imposed by the operating environment, the task of dealing with potential hash collisions is difficult. One approach is to have new flows preempt previous flows when hash table collisions occur. The benefit of this processing behavior is that the system can quickly respond to hash table lookups because each hash bucket only contains one entry. With a total of 8 million hash buckets, the frequency of hash collisions would be low. The downside of this approach is that interesting flows may not be fully monitored because state information for an active flow is lost when a hash table collision occurs.

Another approach is to support a linked list of flow state entries tied to each hash bucket. One advantage with this solution is that all flows are monitored, regardless of whether or not there were hash table collisions. One down side is that it may take an excessive amount of time to retrieve flow state information from the state store because the state store manager may have to traverse a long linked list of entries. This delay in retrieving state information can lead to data loss on the network device which will adversely affect the overall throughput of the network.

[00121] In order to improve the performance of the system, herein described engine will not perform any memory operations to the state store when processing a TCP SYN packet. Instead, these packets are passed through the system without incurring any of the delays associated with an attempt to retrieve state information. In the presence of a TCP SYN attack, the system will pass traffic through without consuming flow state resources. Other non-TCP traffic will also flow through the system without any additional processing.

[00122] The tracking of a flow is initiated by the reception of a TCP data packet. The assumption here is that a proper TCP flow setup has previously been performed by the connection endpoints. A denial of service attack which generates random TCP data packets without first establishing a valid TCP session can potentially induce processing delays for the proposed monitoring system. The flow state manager allocates resources and attempts to track these packets as if they were part of a valid TCP flow. An attack of this nature could potentially exhaust the per-flow state storage resources of the solution.

[00123] There are several methods which could be employed to age flows out of the active flow cache. First, one could set a maximum idle period. If no traffic is detected on a particular flow for a predefined unit of time, then the flow will assumed to have been terminated and the resources that were used to monitor the flow-will-be released. Secondly, a least recently used algorithm is implemented. Instead of aging out flows after a set period of time, the age out of idle flows only

occurs after all of the system resources have been utilized. When a new flow arrives and there are no flow tracking resources available. The resources associated with the flow which has been idle for the longest period of time will be used to support the tracking of the newly arrived flow. This approach eliminates the need for periodic background processing to age out flows because a flow age out is triggered by the arrival of a new flow. A third approach involves cannibalizing the resources of another flow when resource contention occurs. When using a hash table to store flow state information, a flow would be assumed to be terminated whenever a hash table collision occurred during the arrival of a new flow. One disadvantage of this approach is that two or more active flows which map to the same hash table entry will continually be bumping the other flow from the monitoring system. This will inhibit the ability of the monitoring system to fully monitor these flows. One benefit of this technique over the first two is that of performance. The third algorithm can be implemented quickly and takes a small, bounded amount of time to service each flow. The other two algorithms require extra processing in order to maintain link lists of least recently used flows. In addition, the traversal of long link list chains may be required in order to navigate to the proper flow record. This extra processing can cause excessive delays and leads to systems which are prone to data loss. All three of these options have limitations. The modular design of the herein described monitoring engine allows the replacement of the State Store Manager component. All of the logic necessary to implement one of these algorithms will be contained within this module on an FPGA. By replacing this module, the behavior of the memory manager can be altered in order to match the behavior of the system with the expected traffic load.

[00124] During any TCP conversation, a situation called overlapping retransmissions can arise. While the occurrence of condition is normal behavior for the TCP protocol, it can cause problems when performing flow reconstruction if not handled properly. To accommodate an overlapping retransmission, the herein described circuit design employs a data enable signal and a valid bytes vector. The

data enable signal will during a clock cycle where there is TCP data to be processed by the client application. Valid bytes is a 4-bit vector which indicates which of the four data bytes contain valid data to be processed. The client application will only process data when both the data enable signal and the appropriate valid bytes signal are asserted. An example of an overlapping retransmission and the controlling signals can be seen in Figure 14.

[00125] The herein described systems and methods that enable content based routing algorithms which support fine grain routing of network packets based on packet payloads, intrusion detection systems which offer a wide range services from the triggering of alarms to packet filtering to virus removal, advanced traffic filtering systems which filter copyrighted or confidential material based on data signatures or watermarks, real-time monitoring systems which operate at multi-gigabit line speeds, data scrubbing system which remove selected content providing enhanced levels of security, and data mining systems which collect data for specialized analysis systems. Extensible networking systems provide a flexible platform for performing these complex tasks. With the continued increase of clock frequencies, gate counts, and memory densities in microprocessors and Field Programmable Gate Arrays (FPGA), vast amounts of hardware resources can be made available to the extensible networking solutions developers. Instead of just forwarding packets, new network devices will be able to provide value added services within the core of the Internet. A hardware circuit which supports TCP stream re-assembly and flow monitoring is a desired component which will allow these services to operate in a high speed networking environment.

[00126] Also, the herein described systems and methods can be used to police copyrights, which is one technical effect. System 100 can be keyed with a data pattern that will reliably detect when a party's copyrighted material is transmitted over a network. For example, copyrighted songs, motion pictures, and images are often transmitted over the Internet via audio files, video files, and image files. By

properly designing a data pattern that will detect when such works are present in packet traffic, a practitioner of the herein described systems and methods can utilize system 100 to detect the transmission of such copyrighted works and take appropriate action upon detection.

[00127] Further still, the herein described systems and methods can be used to protect against the dissemination of trade secrets and confidential documents, which is another technical effect. A company having trade secrets and/or confidential documents stored on its internal computer system can utilize the herein described systems and methods to prevent the unauthorized transmission of such information outside a company's internal network. The company's network firewall can use system 100 that is keyed to detect and drop any unauthorized packets that are found to include a string that matches a data pattern that encompasses that company's trade secrets and/or confidential information. A company has a wide range of options for flagging their confidential/trade secret information, from adding electronic watermarks to such information (wherein the data processor is keyed by the watermark) to designing a separate data pattern for each confidential/trade secret document/file that will reliably detect when that document/file is transmitted.

[00128] Further still, the herein described systems and methods can be utilized by governmental investigatory agencies to monitor data transmissions of targeted entities over a computer network, which is another technical effect. System 100 can be keyed with a data pattern that encompasses keywords of interest and variations thereof. For example, certain words related to explosives (i.e., TNT, etc.), crimes (i.e., kill, rob, etc.), and/or wanted individuals (i.e., known terrorists, fugitives, etc.) can be keyed into the packet processor. Once so configured, the packet processor can detect whether those keywords (or variations) are present in a packet stream, and upon detection take appropriate action (e.g., notify an interested governmental agency, or redirect the data for further automated processing).

[00129] Yet another example of an application for the herein described systems and methods is as a language translator, which is another technical effect. System 100's search and replace capabilities can be used to detect when a word in a first language is present in a packet, and upon detection, replace that word with its translation into a second language. For example, the packet processor can be used to replace the word "friend" when detected in a packet with its Spanish translation "amigo". Taking advantage of the fact that system 100 possesses the capability of searching packets for a plurality of different data patterns, the present invention can be used as a large scale translation device wherein the packet processor is keyed with a large language A to language B dictionary. Further still, it is possible that a practitioner of the herein described systems and methods can develop data patterns that not only take into account word-for-word translations, but also will account for grammatical issues (for example, to reconcile the English method of a noun preceded by an adjective with the Spanish method of a noun followed by an adjective).

[00130] Further still, the herein described systems and methods can be used to monitor/filter packet traffic for offensive content, which is another technical effect. For example, a parent may wish to use system 100 to prevent a child from receiving profane or pornographic material over the Internet. By keying system 100 to search for and delete profanities or potentially pornographic material, a parent can prevent such offensive material from reaching their home computer.

[00131] Yet another potential application is as an encryption/ decryption device, which is yet another technical effect. System 100 can be designed to replace various words or letters with replacement codes to thereby encrypt packets designed for the network. On the receiving end, another System 100 can be equipped to decrypt the encrypted packets by replacing the replacement codes with the original data.

[00132] These are but a few of the potential uses and technical effects of the herein described methods and systems. Those of ordinary skill in the art

will readily recognize additional uses for the present invention, and as such, the scope of the present invention should not be limited to the above-described applications which are merely illustrative of the wide range of usefulness possessed by the present invention. The full scope of the present invention can be determined upon review of the description above and the attached claims.

[00133] While the invention has been described in terms of various specific embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the claims.

WHAT IS CLAIMED IS:

1. A method for obtaining data while facilitating keeping a minimum amount of state, said method comprising:

receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device;

removing an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame with the logic device;

determining a validity of a checksum of the removed steam-oriented protocol header;

dropping the IP frame when the checksum is invalid;

actively dropping IP frames such that the first device always has an accumulated in order content stream before the third device accumulates the in order stream;

supplying a client application with data from the removed protocol frame when the checksum is valid; and

sending an IP frame including the removed stream-oriented protocol frame to the third device from the first device when the checksum is valid.

2. A method in accordance with Claim 1 wherein sending an IP frame including the removed protocol frame comprises:

classifying the removed protocol frame as at least one of a sequence number greater than expected classification, a sequence number equal to expected classification, and a sequence number less than expected classification;

sending an IP frame including the removed protocol frame to the third device from the first device when the classification is one of the sequence number less than expected classification and the sequence number equal to expected classification; and

dropping an IP frame including the removed protocol frame when the classification is the sequence number greater than expected classification.

3. A method in accordance with Claim 1 wherein removing an embedded stream-oriented protocol frame from the received IP frame with the logic device comprises removing a Transmission Control Protocol (TCP) frame from the received IP frame with the logic device

4. A method in accordance with Claim 1 further comprising classifying the removed embedded stream-oriented protocol frame as at least one of a sequence number greater than expected classification, a sequence number less than expected classification, an invalid Transmission Control Protocol (TCP) checksum classification, a non-TCP classification, a TCP synchronization (TCP SYN) classification, and an else classification.

5. A method in accordance with Claim 4 wherein supplying a client application with data from the removed protocol frame comprises supplying a client application with data from the removed protocol frame only when the classification is else.

6. A method in accordance with Claim 5 wherein sending an IP frame including the removed protocol frame to the third device from the first device comprises sending an IP frame including the removed protocol header to the third device from the first device only when the classification is at least one of non-TCP, TCP SYN, sequence number less than expected, and else.

7. A method in accordance with Claim 6 further comprising dropping the received IP frame when the classification is at least one of invalid TCP checksum and sequence number greater than expected.

8. A method in accordance with Claim 1 wherein supplying a client application with data from the removed protocol frame comprises supplying a client application with data from the removed protocol frame such that content of a Transmission Control Protocol (TCP) stream is provided to a client application in order.

9. A method in accordance with Claim 1 wherein sending an IP frame including the removed protocol frame to the third device from the first device comprises sending the IP frame received by the first device from the first device to the third device.

10. A method in accordance with Claim 1 wherein supplying a client application with data from the removed protocol frame comprises supplying a client application that counts bits with data from the removed protocol frame.

11. A method in accordance with Claim 1 wherein supplying a client application with data from the removed protocol frame comprises supplying a client application with data from the removed protocol frame wherein the client application compares the supplied data with reference data to perform content matching.

12. A method in accordance with Claim 1 wherein supplying a client application with data from the removed protocol frame when the checksum is valid comprises supplying a client application with data from the removed protocol frame comprising FPGA programming data.

13. An apparatus for facilitating keeping a minimum amount of state, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

receive an Internet Protocol (IP) frame sent from a first device addressed to a second device;

remove an embedded stream-oriented protocol frame including a header and data packet from the received IP frame;

classify the removed protocol frame as at least one of a sequence number greater than expected classification, a sequence number equal to expected classification, and a sequence number less than expected classification;

send an IP frame including the removed protocol frame to the second device when the classification is one of the sequence number less than expected classification and the sequence number equal to expected classification;

drop the received IP frame including the removed protocol frame when the classification is the sequence number greater than expected classification; and

actively drop IP frames such that said apparatus always has an accumulated in order content stream before the second device accumulates the in order stream.

14. An apparatus in accordance with Claim 13 wherein said logic device further configured to:

remove a Transmission Control Protocol (TCP) frame from the received IP frame;

determine a validity of a TCP checksum of the removed TCP frame;

drop the IP frame when the TCP checksum is invalid; and

send an IP frame including the removed TCP frame to the second device when the TCP checksum is valid.

15. An apparatus in accordance with Claim 13 wherein said logic device further configured to classify the removed protocol frame as at least one of a sequence number greater than expected classification, a sequence number less than expected classification, an invalid TCP checksum classification, a non-TCP classification, a TCP synchronization (TCP SYN) classification, and an else classification.

16. An apparatus in accordance with Claim 15 wherein said logic device further configured to supply a client application with data from the removed protocol frame only when the classification is the else classification.

17. An apparatus in accordance with Claim 15 wherein said logic device further configured to send an IP frame including the removed protocol frame to the second device only when the classification is at least one of the non-TCP classification, the TCP SYN classification, the sequence number less than expected classification, and the else classification.

18. An apparatus in accordance with Claim 15 wherein said logic device further configured to drop the received IP frame when the classification is at least one of the invalid TCP checksum classification and the sequence number greater than expected classification.

19. An apparatus in accordance with Claim 13 wherein said logic device further configured to supply a client application with data from the removed protocol frame such that content of a Transmission Control Protocol (TCP) stream is provided to a client application in order.

20. An apparatus in accordance with Claim 13 wherein said logic device further configured to send to the second device an IP frame comprising the IP frame received by said IC.

21. An apparatus in accordance with Claim 13 wherein said logic device further configured to supply a client application that counts bits with data from the removed protocol frame.

22. An apparatus in accordance with Claim 13 wherein said logic device further configured to supply a client application with data from the removed protocol frame wherein the client application compares the supplied data with reference data to perform content matching.

23. An apparatus in accordance with Claim 13 further comprising a Field Programmable Gate Array (FPGA) operationally coupled to said logic device, said logic device configured to reprogram said FPGA.

24. An apparatus in accordance with Claim 23 wherein said FPGA operationally coupled to said input port and said output port.

25. An apparatus comprising:

at least one input port;

at least one output port; and

at least one reprogrammable device;

at least one logic device operationally coupled to said input port, said output port, and said reprogrammable device, said logic device configured to:

receive an Internet Protocol (IP) frame sent from a first device addressed to a second device;

remove an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame;

determine if the removed protocol frame includes programming data;

reprogram said reprogrammable device when the removed protocol frame contains programming data; and

send an IP frame including the removed protocol frame to the second device.

26. An apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

receive an Internet Protocol (IP) frame sent from a first device addressed to a second device;

remove an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame;

determine if the removed protocol frame includes data representing at least part of a quality of service (QoS) algorithm;

supply a client application with data from the removed protocol frame when the removed protocol includes data representing at least part of a QoS algorithm; and

send an IP frame including the removed protocol frame to the second device.

27. An apparatus in accordance with Claim 26 wherein said logic device further configured to reconfigure at least one QoS policy in accordance with the QoS data supplied to the client application.

28. A network comprising a plurality of switching devices operationally coupled to each other, at least some of said switching devices comprising at least one logic device configured to:

monitor stream-oriented network traffic for Field Programmable Gate Array (FPGA) programming data;

reprogram at least one of itself and a FPGA coupled to said logic device upon receipt of the FPGA programming data; and

retransmit the FPGA programming data back onto the network such that other switching devices can reprogram at least one of themselves and an attached FPGA using the FPGA programming data.

29. A network in accordance with Claim 28 wherein to monitor stream-oriented network traffic for FPGA programming data, at least some of said switching devices configured to:

receive an Internet Protocol (IP) frame sent from a first device addressed to a second device;

remove an embedded stream-oriented protocol frame from the received IP frame; and

determine if the removed protocol frame includes FPGA programming data.

30. A network in accordance with Claim 28 wherein to monitor stream-oriented network traffic for FPGA programming data, at least some of said switching devices configured to:

receive an Internet Protocol (IP) frame sent from a first device addressed to a second device;

remove an embedded stream-oriented protocol frame from the received IP frame;

determine if the removed steam-oriented protocol frame includes FPGA programming data;

supply a client application with data from the removed protocol frame when the removed protocol includes FPGA programming data; and

send an IP frame including the removed protocol frame to the second device.

31. A method for distributing data, said method comprising

receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including an Integrated Circuit (IC);

removing an embedded protocol frame from the received IP frame with the IC;

supplying a client application with data from the removed protocol frame;

analyzing the data supplied to the client application; and

sending an IP frame including the removed protocol frame to the third device from the first device only after analyzing the data.

32. A method for distributing data on a network using a single TCP/IP source, a single destination and one or more intermediate hardware based monitoring nodes, said method comprising:

receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including a logic device;

removing an embedded Transmission Control Protocol (TCP) frame from the received IP frame with the logic device;

supplying a client application with data from the removed protocol frame; and

sending an IP frame including the removed protocol frame to the third device from the first device after performing an analysis on the removed frame.

33. A method for identifying and selectively removing data on a data transmission system, said method comprising:

receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including a logic device;

actively dropping IP frames addressed to the third device sent by the second device such that the first device always has an accumulated in order content stream before the third device accumulates the in order content stream.

34. A dynamically reconfigureable data transmission system, having an apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

receive an Internet Protocol (IP) frame sent from a first device addressed to a second device;

remove an embedded protocol frame from the received IP frame;

determine if the removed protocol frame includes Field Programmable Field Array (FPGA) programming data;

supply a client application with data from the removed protocol frame when the removed protocol includes FPGA programming data such that the application receives a content stream in order; and

send an IP frame including the removed protocol frame to the second device.

35. A method for controlling traffic on a network, said method comprising:

monitoring a data stream;

determining a particular byte offset within the monitored stream at which to block flow of the stream; and

blocking flow of the data stream at the determined byte offset.

36. A method in accordance with Claim 35 further comprising:

sending data to an authorizing authority; and

re-enabling flow of the blocked stream upon receipt of an authorization from the authorizing authority.

37. A method in accordance with Claim 35 further comprising manipulating the data stream such that a second device comprising a receiving device receives data different than that sent from a first device comprising a sending device.

38. A method in accordance with Claim 37, wherein said monitoring a data stream comprises monitoring TCP traffic in band through a switch using a plurality of content scanning engines.

39. A method in accordance with Claim 38, wherein said monitoring comprises content scanning a plurality of TCP packets to detect a content match that spans multiple packets.

40. A method in accordance with Claim 39, wherein said monitoring comprises monitoring a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

41. A method in accordance with Claim 40 further comprising:

maintaining a period of idle time for each monitored flow; and

stopping monitoring a flow which has the longest period of idle time upon receipt of a new flow to be monitored when a total number of flows being monitored is equal to a desired maximum number of simultaneous flows.

42. A method in accordance with Claim 40, wherein said monitoring a plurality of data flows comprises monitoring a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry, said method further comprises:

receiving a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision; and

stopping monitoring of the existing flow whose hash table entry the new flow collided with.

43. A method in accordance with Claim 35, wherein said monitoring comprises monitoring a TCP data stream for a predetermined condition, said blocking comprises generating and transmitting a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

44. A method for controlling traffic on a network, said method comprising:

monitoring a data stream for a first predetermined condition;

blocking flow of the data steam upon a detection of the first predetermined condition; and

re-enabling flow of the blocked stream.

45. A method in accordance with Claim 44 further comprising:

sending data to an authorizing authority; and

re-enabling flow of the blocked stream upon receipt of an authorization from the authorizing authority.

46. A method in accordance with Claim 45, wherein said monitoring comprises monitoring a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

47. A method for controlling traffic on a network, said method comprising:

monitoring a TCP data stream for a predetermined condition; and

generating and transmitting a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

48. A method for controlling traffic on a network, said method comprising:

monitoring a TCP data stream from a first device directed toward a second device for a predetermined condition; and

manipulating the TCP data stream such that the second device receives data different than that sent from the first device.

49. A method in accordance with Claim 48, wherein said monitoring comprises monitoring a TCP data stream from a first device directed toward a second device for an indication of a presence of a virus within the stream, said manipulating comprises removing the virus from the stream.

50. A method in accordance with Claim 48, wherein said manipulating comprises computing a checksum for a modified TCP packet which is the same number of bytes long as an original TCP packet which the modified TCP packet is replacing in the TCP data stream.

51. A method in accordance with Claim 48, wherein said manipulating comprises computing a checksum for a TCP packet added to the stream.

52. A method for controlling traffic on a network, said method comprising monitoring TCP traffic in band through a switch using a plurality of content scanning engines.

53. A method in accordance with Claim 52 further comprising buffering the monitored TCP traffic in an input buffer.

54. A method in accordance with Claim 52 further comprising validating and classifying TCP packets as part of individual flows and sending the flows to the content scanning engines such that each content scanning engine receives a unique flow.

55. A method in accordance with Claim 54, wherein said classifying comprises using a hash table to classify a flow wherein hash table conflicts are resolved by chaining off a linked list of flow state records wherein a length of the chain is limited to a constant number of entries.

56. A method for controlling traffic on a network, said method comprising content scanning a plurality of TCP packets to detect a content match that spans multiple packets.

57. A method for controlling traffic on a network, said method comprising monitoring a plurality of flows through the network wherein per flow memory usage is matched to a burst width of a memory module used to monitor a flow.

58. A method for controlling traffic on a network, said method comprising monitoring a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

59. A method for controlling traffic on a network said method comprising:

monitoring a plurality of data flows simultaneously;

assigning a maximum idle period of time for each monitored flow; and

stopping monitoring a flow which is idle for at least the assigned period of time.

60. A method for controlling traffic on a network said method comprising:

monitoring a plurality of data flows simultaneously;

maintaining a period of idle time for each monitored flow; and

stopping monitoring the flow having a longest period of idle time.

61. A method in accordance with Claim 60, wherein said stopping comprises stopping monitoring a flow having the longest period of idle time upon receipt of a new flow to be monitored.

62. A method in accordance with Claim 60, wherein said stopping comprises stopping monitoring a flow which has the longest period of idle time upon receipt of a new flow to be monitored when a total number of flows being monitored is equal to a desired maximum number of simultaneous flows.

63. A method for controlling traffic on a network said method comprising:

monitoring a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry;

receiving a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision; and

stopping monitoring of the existing flow whose hash table entry the new flow collided with.

64. A method in accordance with Claim 63 further comprising monitoring the new flow after said stopping monitoring the existing flow.

65. A Field Programmable Gate Array (FPGA) configured to:

monitor a plurality of data flows using a hash table to store state information regarding each flow;

resolve hash table collisions according to a first algorithm stored on said FPGA;

receive a second algorithm at said FPGA to resolve hash table collisions, said second algorithm different from the first algorithm; and

use the received second algorithm to resolve hash table collisions occurring subsequent said receipt of the second algorithm.

66 An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a data stream;

determine a particular byte offset within the monitored stream at which to block flow of the stream; and

block flow of the data stream at the determined byte offset.

67. An apparatus in accordance with Claim 66, wherein said logic device further configured to:

send data to an authorizing authority; and

re-enable flow of the blocked stream upon receipt of an authorization from the authorizing authority.

68. An apparatus in accordance with Claim 66, wherein said logic device further configured to manipulate the data stream such that a second device comprising a receiving device receives data different than that sent from a first device comprising a sending device.

69. An apparatus in accordance with Claim 66, wherein said logic device further configured to monitor TCP traffic in band using a plurality of content scanning engines.

70. An apparatus in accordance with Claim 66, wherein said logic device further configured to content scan a plurality of TCP packets to detect a content match that spans multiple packets.

71. An apparatus in accordance with Claim 66 , wherein said logic device further configured to monitor a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

72. An apparatus in accordance with Claim 66, wherein said logic device further configured to:

maintain a period of idle time for each monitored flow; and

stop monitoring a flow which has the longest period of idle time upon receipt of a new flow to be monitored when a total number of flows being monitored is equal to a desired maximum number of simultaneous flows.

73. An apparatus in accordance with Claim 66, wherein said logic device further configured to:

monitor a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry;

receive a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision; and

stop monitoring of the existing flow whose hash table entry the new flow collided with.

74. An apparatus in accordance with Claim 66, wherein said logic device further configured to:

monitor a TCP data stream for a predetermined condition; and

generate and transmit a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

75. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a data stream for a first predetermined condition;

block flow of the data steam upon a detection of the first predetermined condition; and

re-enable flow of the blocked stream.

76. An apparatus in accordance with Claim 75, wherein said logic device further configured to:

send data to an authorizing authority; and

re-enable flow of the blocked stream upon receipt of an authorization from the authorizing authority.

76. An apparatus in accordance with Claim 75, wherein said logic device further configured to monitor a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

78. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a TCP data stream for a predetermined condition; and

generate and transmit a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

79. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a TCP data stream from a first device directed toward a second device for a predetermined condition; and

manipulate the TCP data stream such that the second device receives data different than that sent from the first device.

80. An apparatus in accordance with Claim 79, wherein said logic device further configured to:

monitor a TCP data stream from a first device directed toward a second device for an indication of a presence of a virus within the stream; and

remove the virus from the stream.

81. An apparatus in accordance with Claim 79, wherein said logic device further configured to compute a checksum for a modified TCP packet which is the same number of bytes long as an original TCP packet which the modified TCP packet is replacing in the TCP data stream.

82. An apparatus in accordance with Claim 79, wherein said logic device further configured to compute a checksum for a TCP packet added to the stream.

83. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor TCP traffic in band using a plurality of content scanning engines.

84. An apparatus in accordance with Claim 83, wherein said logic device further configured to buffer the monitored TCP traffic in an input buffer.

85. An apparatus in accordance with Claim 83, wherein said logic device further configured to validate and classify TCP packets as part of individual flows and send the flows to the content scanning engines such that each content scanning engine receives a unique flow.

86. An apparatus in accordance with Claim 85, wherein said logic device further configured to use a hash table to classify a flow wherein hash table conflicts are resolved by chaining off a linked list of flow state records wherein a length of the chain is limited to a constant number of entries.

87. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to scan a plurality of TCP packets to detect a content match that spans multiple packets.

88. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to monitor a plurality of flows through the network wherein per flow memory usage is matched to a burst width of a memory module used to monitor a flow.

89. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to monitor a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

90. An apparatus for controlling traffic on a network said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a plurality of data flows simultaneously;

assign a maximum idle period of time for each monitored flow; and

stop monitoring a flow which is idle for at least the assigned period of time.

91. An apparatus for controlling traffic on a network said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a plurality of data flows simultaneously;

maintain a period of idle time for each monitored flow; and

stop monitoring the flow having a longest period of idle time.

92. An apparatus in accordance with Claim 91, wherein said logic device further configured to stop monitoring a flow having the longest period of idle time upon receipt of a new flow to be monitored.

93. An apparatus in accordance with Claim 91, wherein said logic device further configured to stop monitoring a flow which has the longest period of idle time upon receipt of a new flow to be monitored when a total number of flows being monitored is equal to a desired maximum number of simultaneous flows.

94. An apparatus for controlling traffic on a network, said apparatus comprising:

at least one input port;

at least one output port; and

at least one logic device operationally coupled to said input port and said output port, said logic device configured to:

monitor a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry;

receive a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision; and

stop monitoring of the existing flow whose hash table entry the new flow collided with.

95. An apparatus in accordance with Claim 94, wherein said logic device further configured to monitor the new flow after stopping monitoring the existing flow.

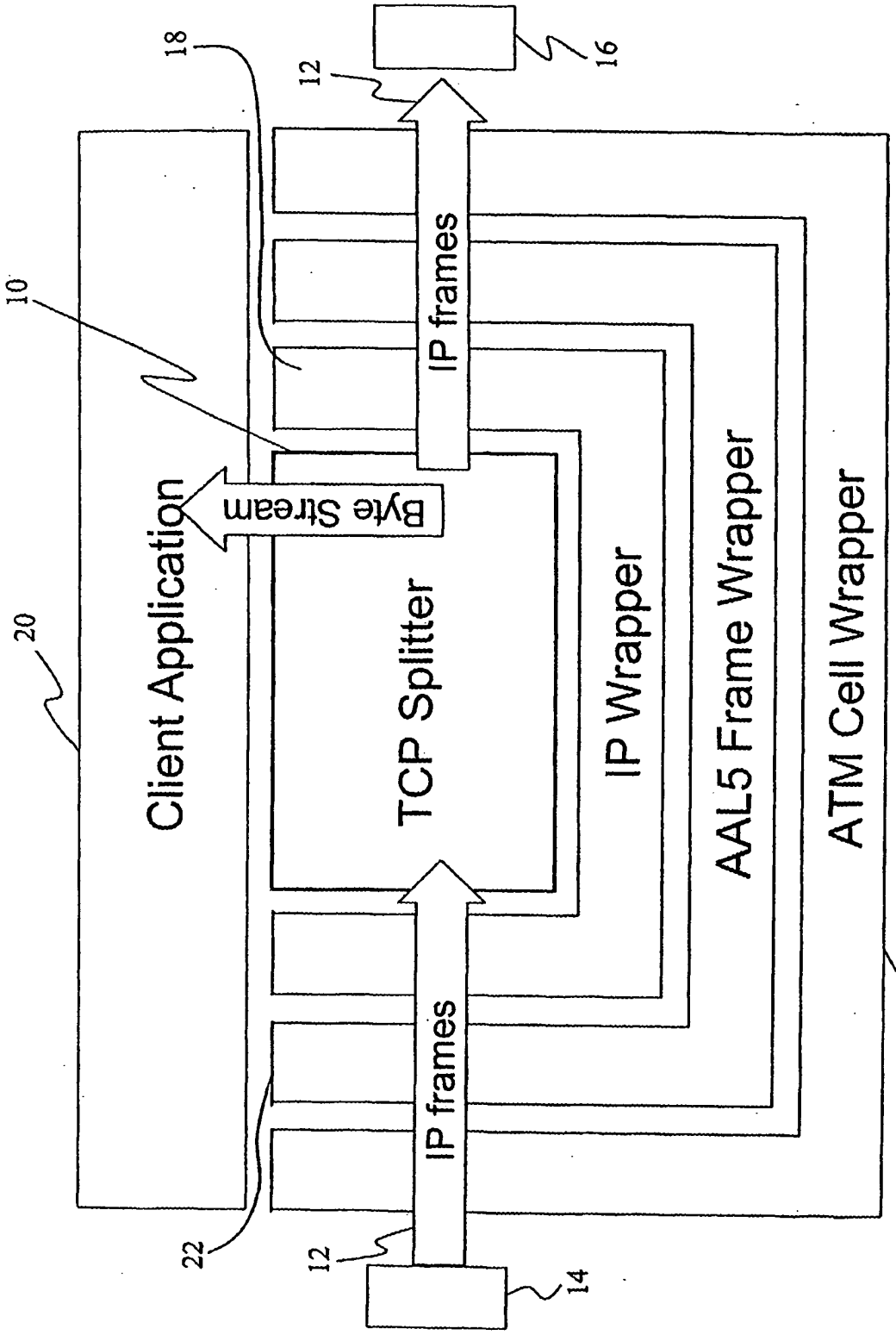

18
12
16
10
IP frames
Byte Stream
Client Application
TCP Splitter
IP Wrapper
AAL5 Frame Wrapper
ATM Cell Wrapper
20
24
22
12
IP frames
14

FIG. 1

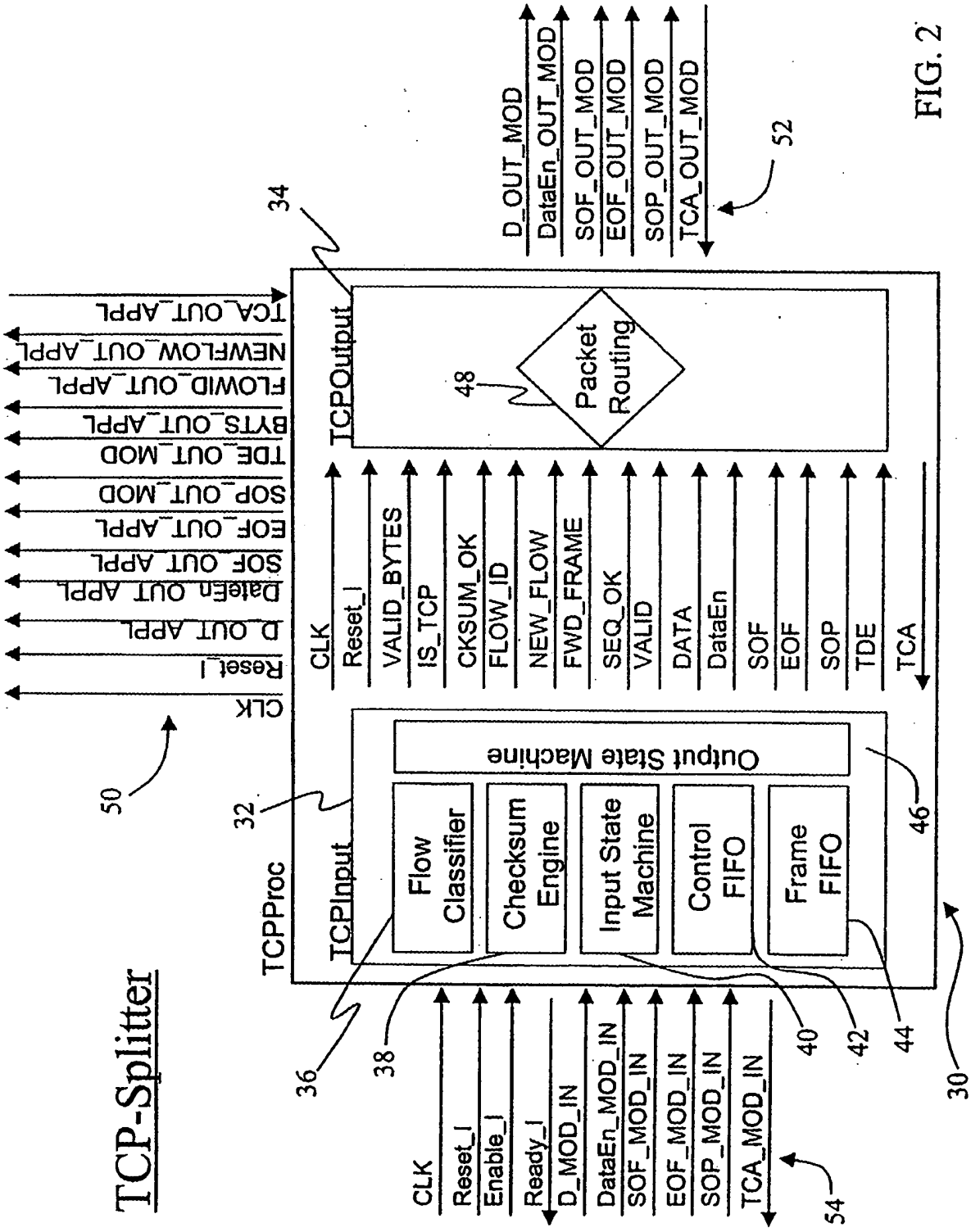

TCP-Splitter
TCPProc
TCPInput
TCPOutput
Flow Classifier
Checksum Engine
Input State Machine
Control FIFO
Frame FIFO
Output State Machine
Packet Routing
CLK
Reset_I
Enable_I
Ready_I
D_MOD_IN
DataEn_MOD_IN
SOF_MOD_IN
EOF_MOD_IN
SOP_MOD_IN
TCA_MOD_IN
D_OUT_MOD
DataEn_OUT_MOD
SOF_OUT_MOD
EOF_OUT_MOD
SOP_OUT_MOD
TCA_OUT_MOD
VALID_BYTES
IS_TCP
CKSUM_OK
FLOW_ID
NEW_FLOW
FWD_FRAME
SEQ_OK
VALID
DATA
DataEn
SOF
EOF
SOP
TDE
TCA
D_OUT_APPL
DataEn_OUT_APPL
SOF_OUT_APPL
EOF_OUT_APPL
SOP_OUT_MOD
TDE_OUT_MOD
BYTS_OUT_APPL
FLOWID_OUT_APPL
NEWFLOW_OUT_APPL
TCA_OUT_APPL
30
32
34
36
38
40
42
44
46
48
50
52
54

FIG. 2

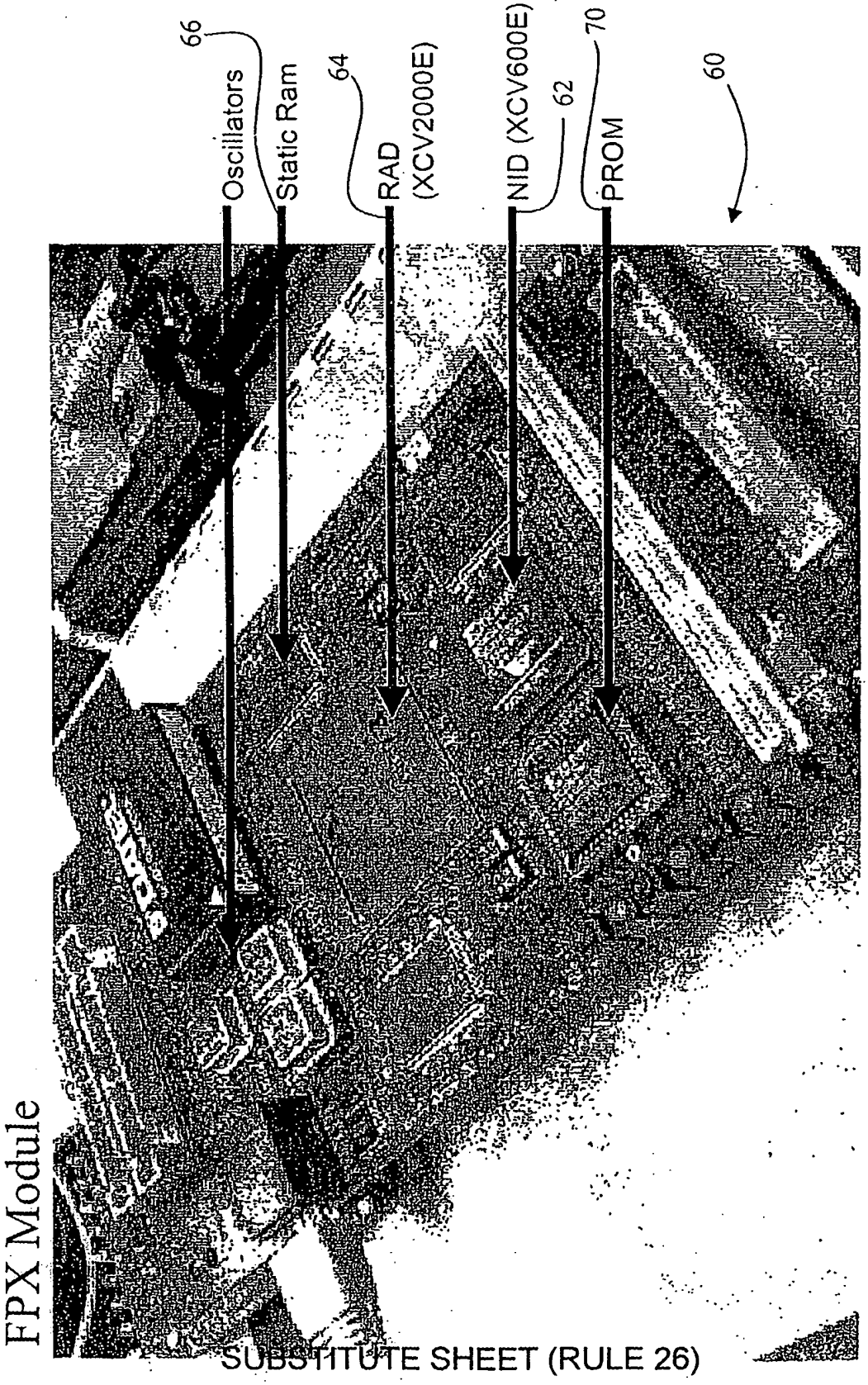

66
Oscillators
Static Ram
64
RAD
(XCV2000E)
NID (XCV600E)
62
70
PROM
60
FPX Module

FIG. 3

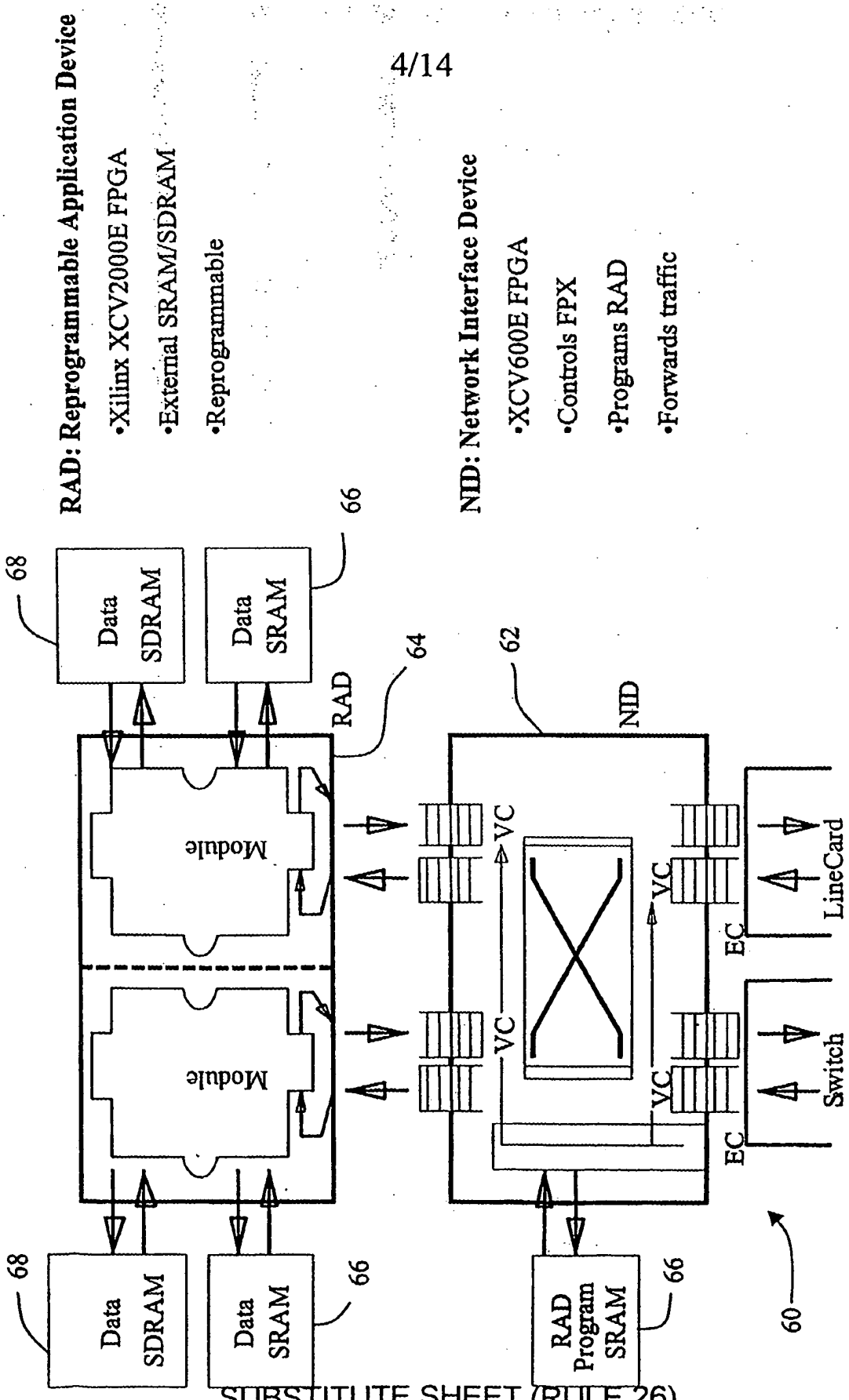

RAD: Reprogrammable Application Device
•Xilinx XCV2000E FPGA
•External SRAM/SDRAM
•Reprogrammable
NID: Network Interface Device
•XCV600E FPGA
•Controls FPX
•Programs RAD
•Forwards traffic
68
Data SDRAM
66
Data SRAM
64
RAD
Module
Module
62
NID
VC
VC
VC
VC
EC
EC
LineCard
Switch
68
Data SDRAM
66
Data SRAM
66
RAD Program SRAM
60

FIG. 4

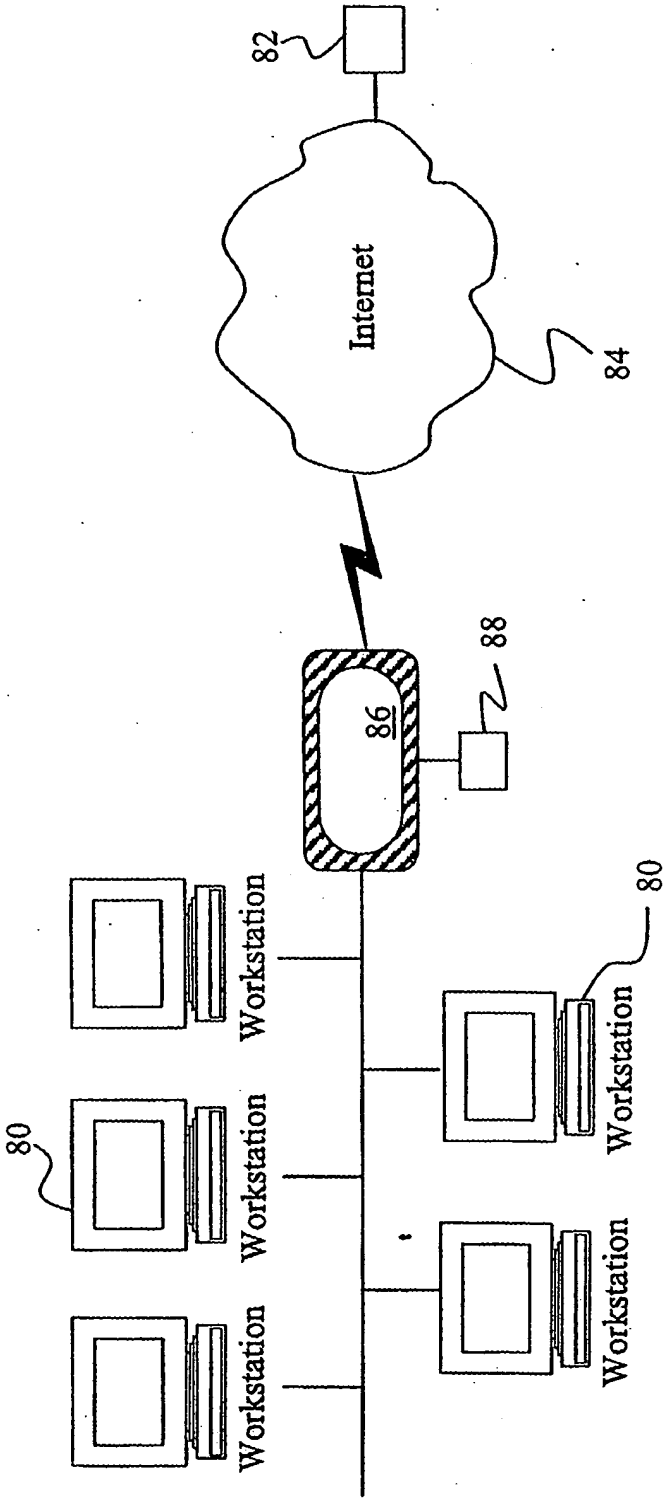

82
Internet
84
86
88
80
Workstation
Workstation
Workstation
80
Workstation
Workstation

FIG. 5

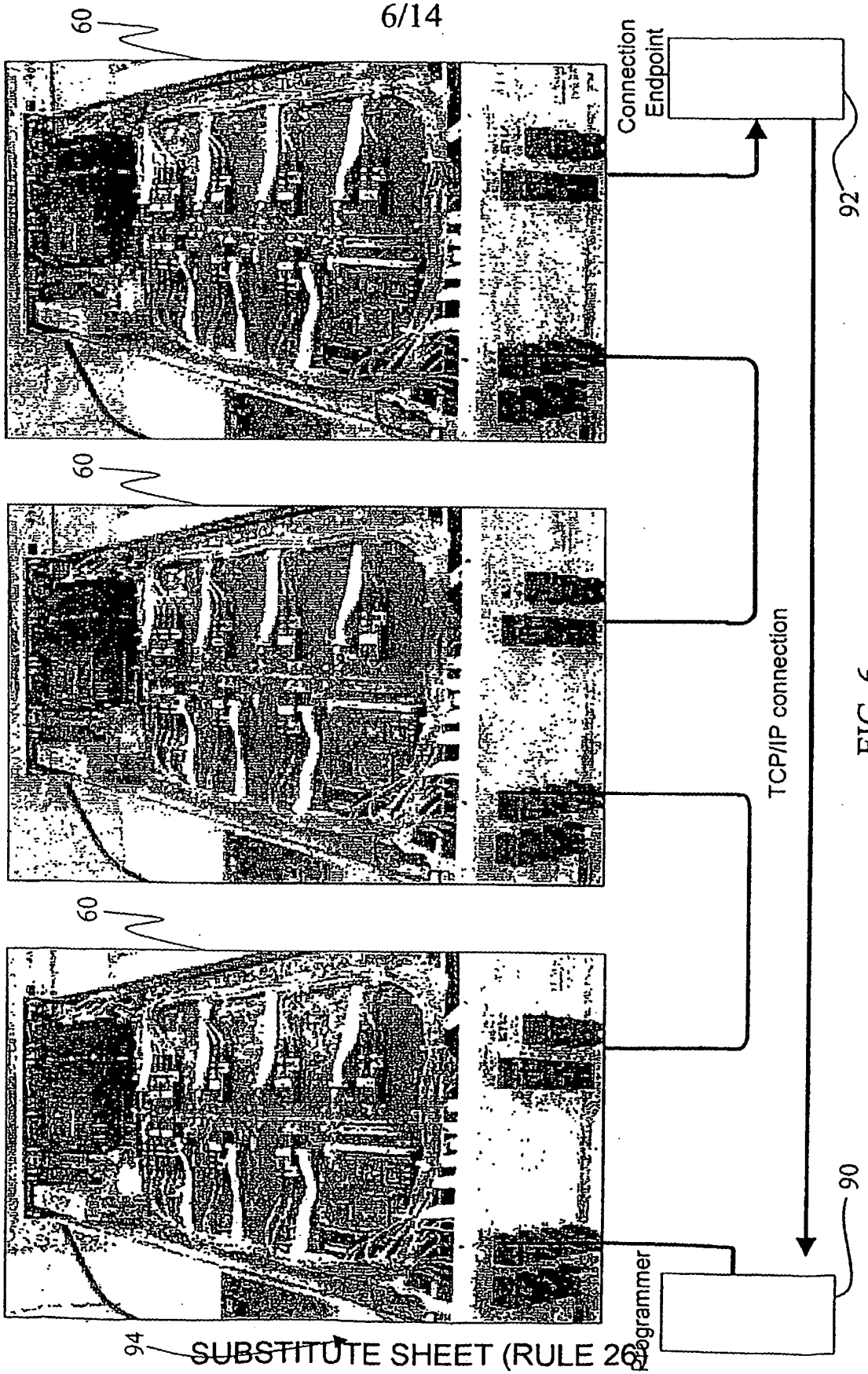

60
60
60
Connection Endpoint
92
TCP/IP connection
90
Programmer
94

FIG. 6

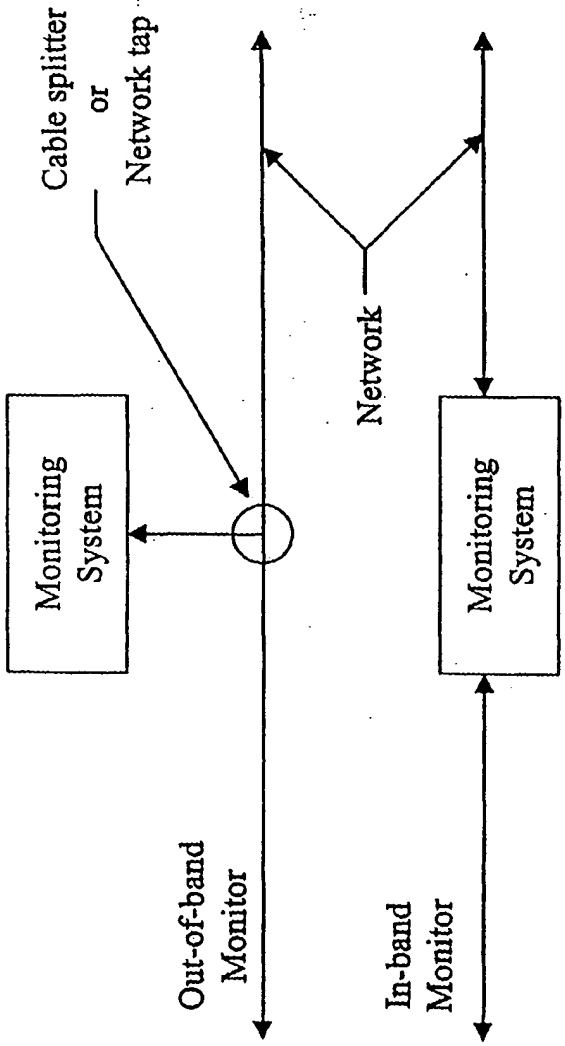


FIG. 7 Types of Monitoring Systems

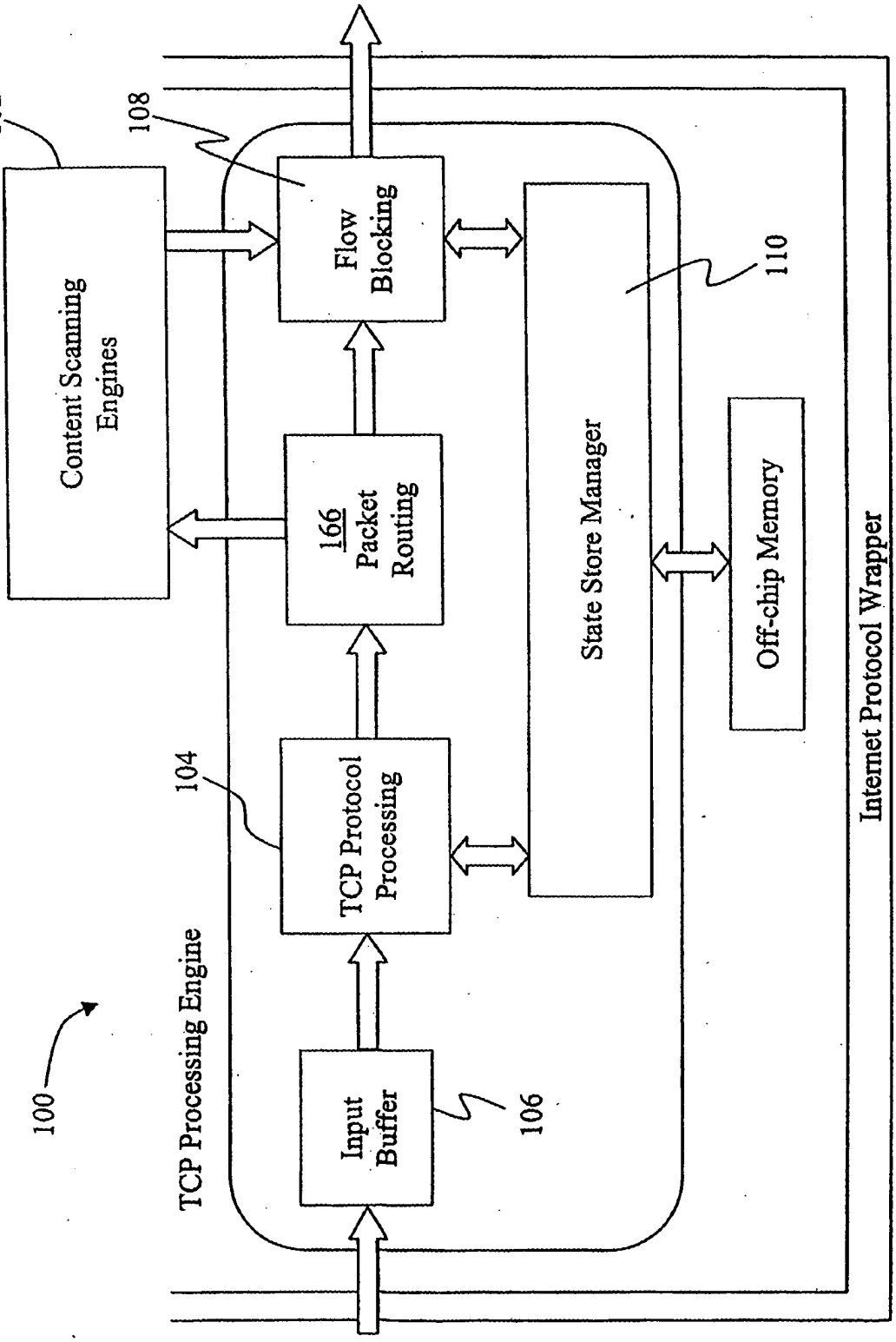

102
108
Content Scanning Engines
Flow Blocking
110
State Store Manager
Off-chip Memory
166 Packet Routing
104
TCP Protocol Processing
Input Buffer
106
TCP Processing Engine
Internet Protocol Wrapper
100

FIG. 8

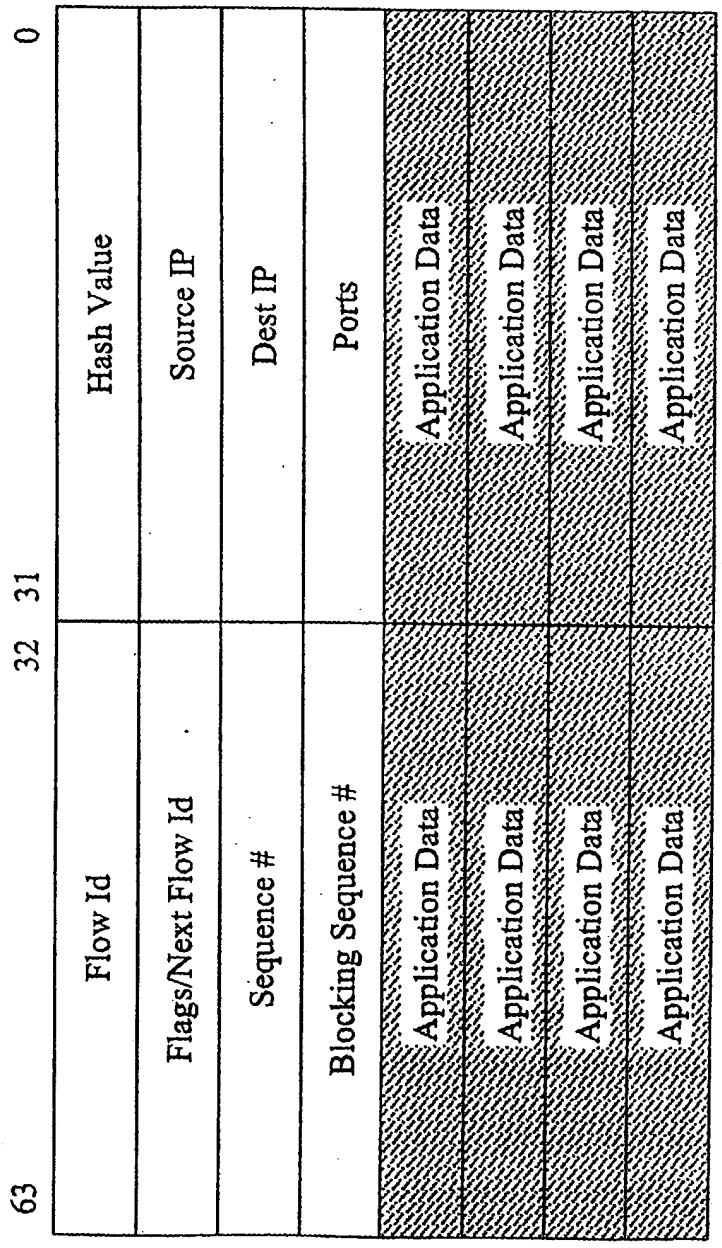

0
31
32
63
Hash Value
Source IP
Dest IP
Ports
Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Flow Id
Flags/Next Flow Id
Sequence #
Blocking Sequence #

FIG. 9

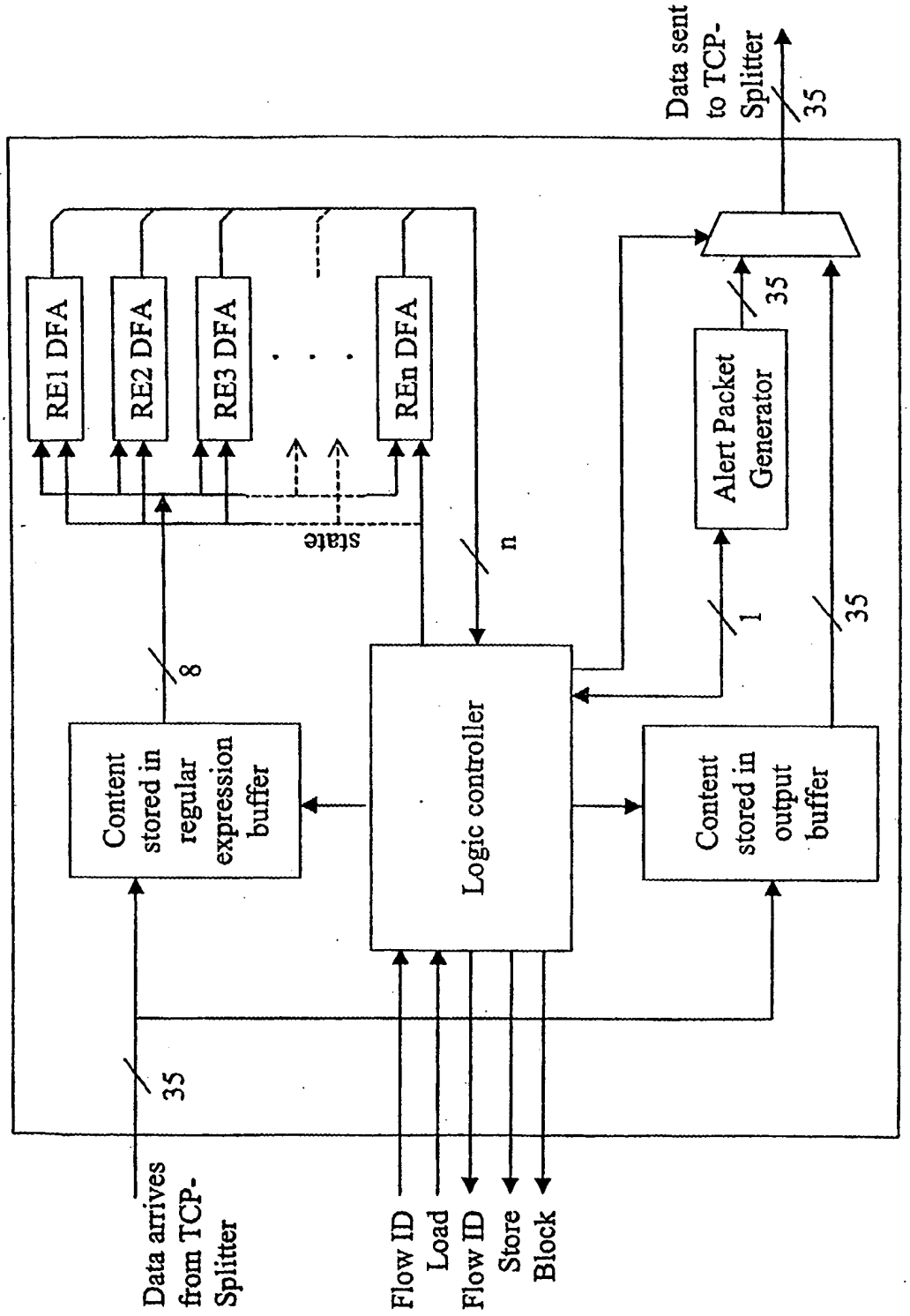

Data sent to TCP-Splitter
35
RE1 DFA
RE2 DFA
RE3 DFA
REn DFA
Alert Packet Generator
35
state
n
8
1
35
Content stored in regular expression buffer
Logic controller
Content stored in output buffer
35
Data arrives from TCP-Splitter
Flow ID
Load
Flow ID
Store
Block

FIG. 10

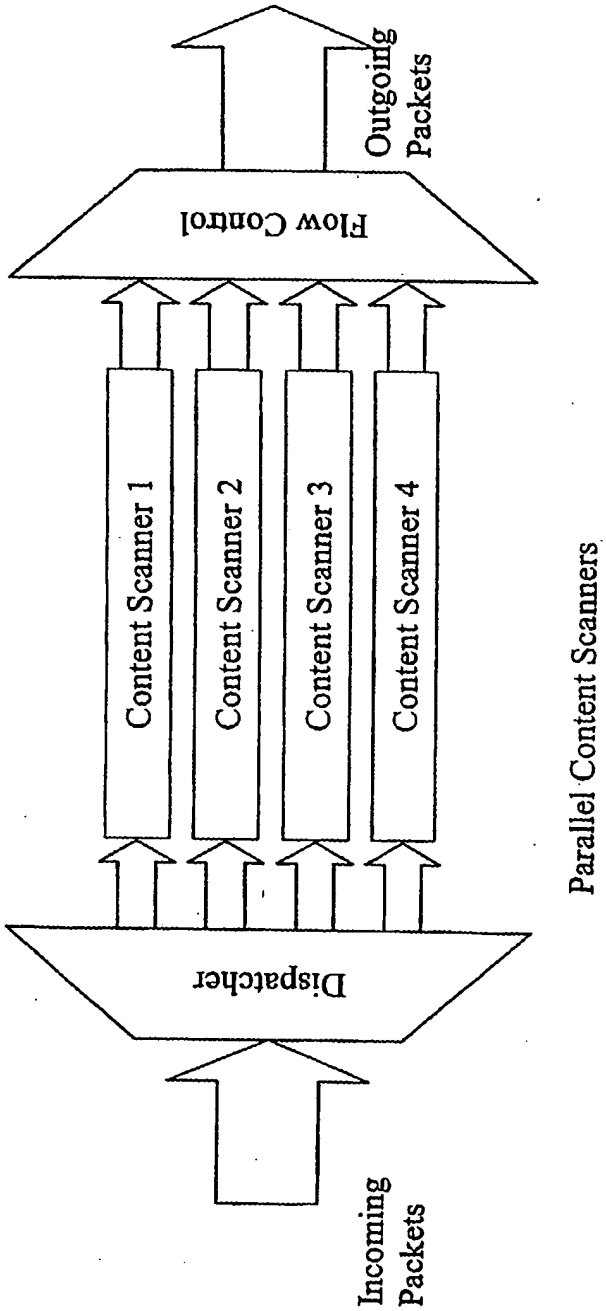

Outgoing Packets
Flow Control
Content Scanner 1
Content Scanner 2
Content Scanner 3
Content Scanner 4
Parallel Content Scanners
Dispatcher
Incoming Packets

FIG. 11

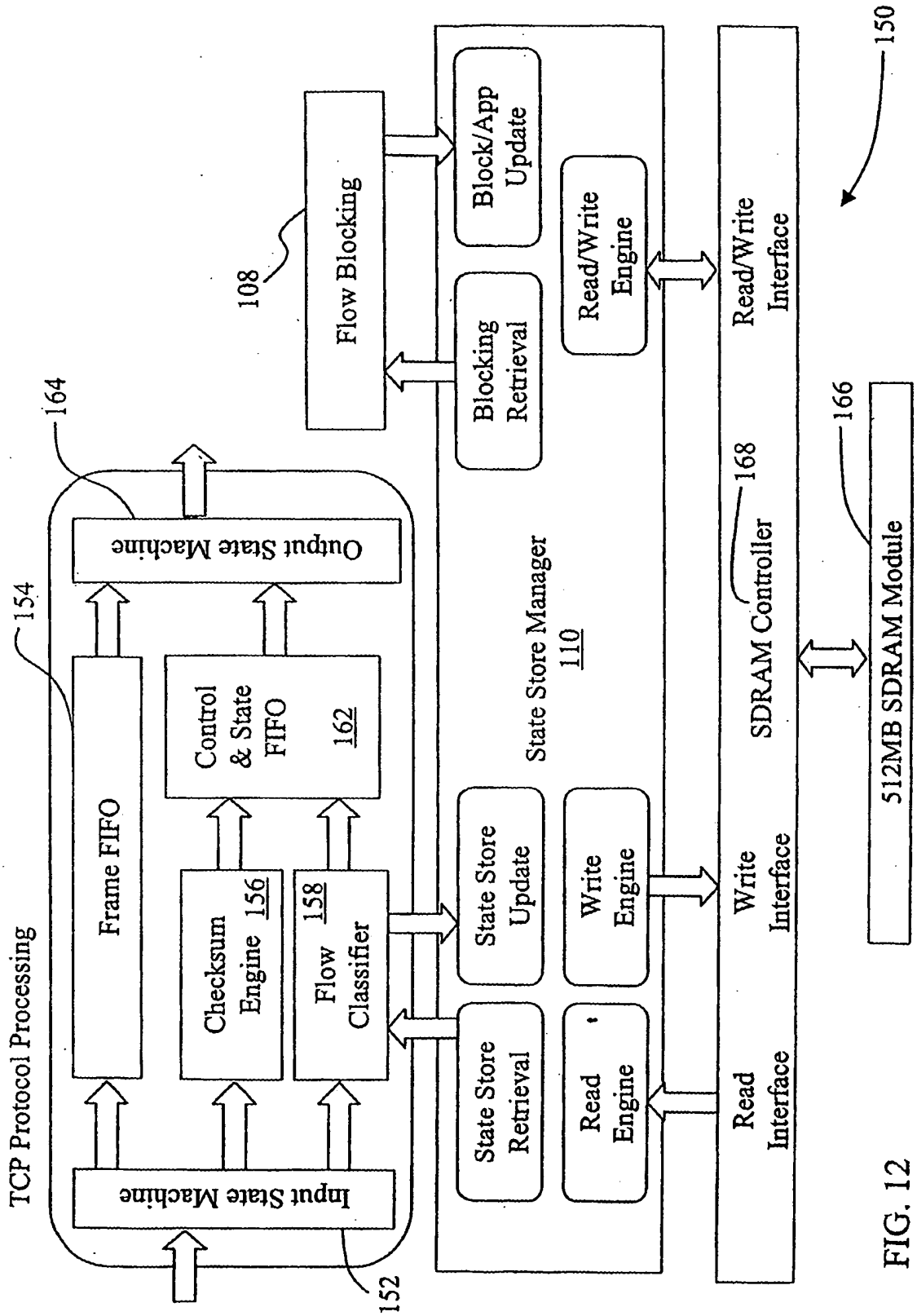

150
Flow Blocking
108
Block/App Update
Blocking Retrieval
Read/Write Engine
Read/Write Interface
164
166
168
Output State Machine
154
Frame FIFO
Control & State FIFO 162
State Store Manager 110
SDRAM Controller
512MB SDRAM Module
Checksum Engine 156
Flow Classifier 158
State Store Update
Write Engine
Write Interface
TCP Protocol Processing
State Store Retrieval
Read Engine
Read Interface
Input State Machine
152

FIG. 12

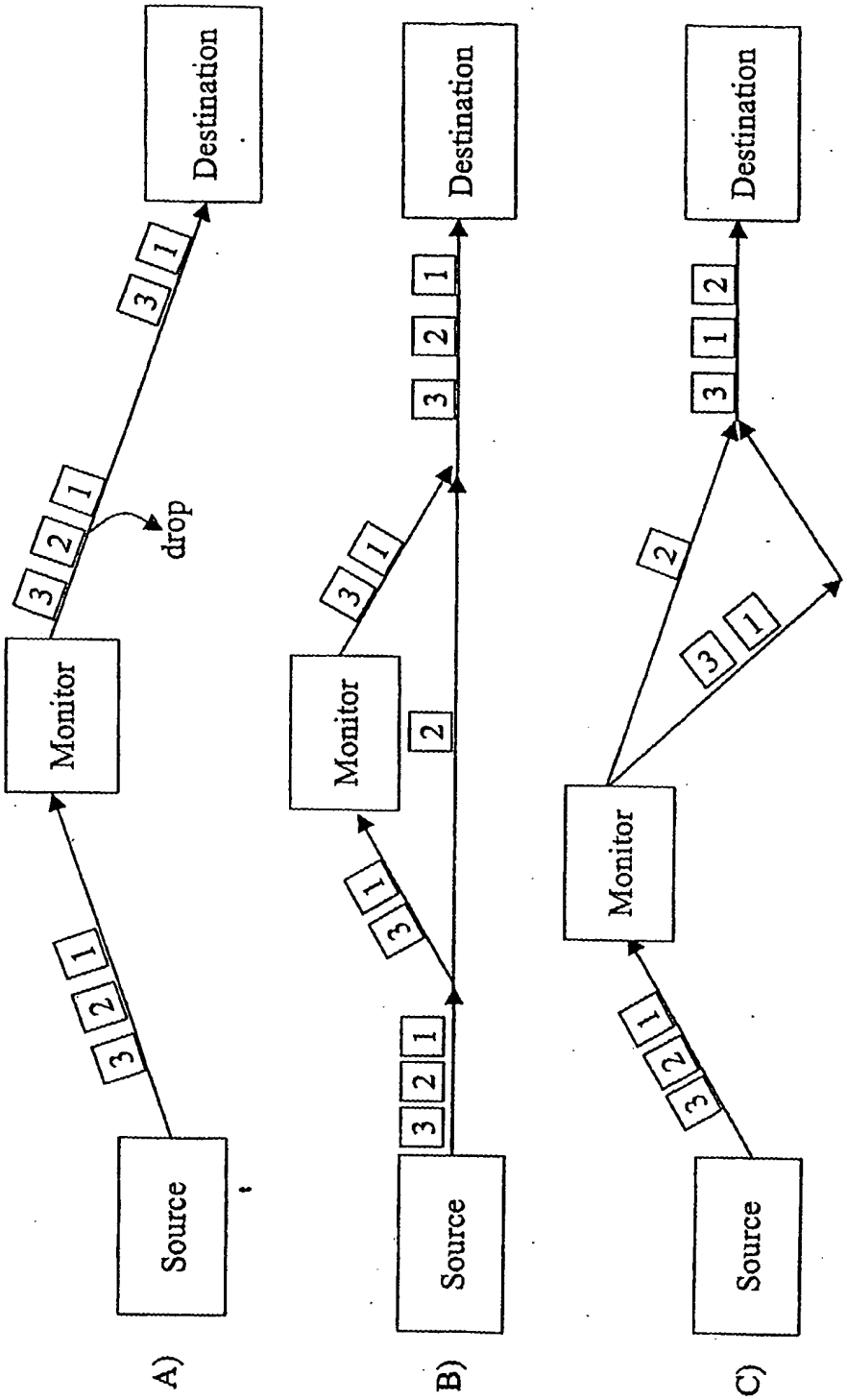


FIG. 13 Packet sequence variations

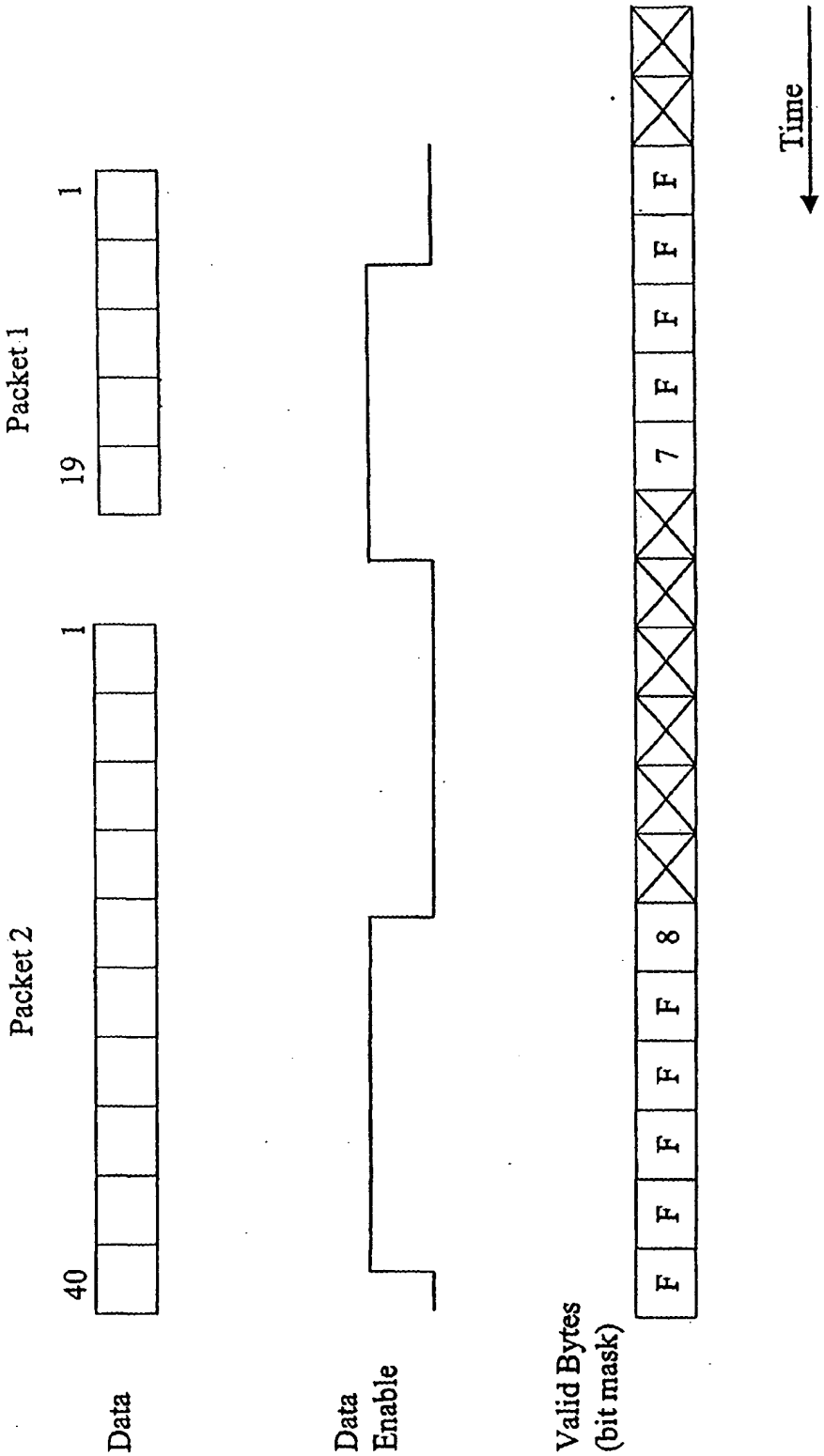


FIG. 14 Overlapping Retransmissions

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 February 2004 (26.02.2004)

PCT

(10) International Publication Number
WO 2004/017604 A3

(51) International Patent Classification[7]: H04L 12/26, 29/06, 12/14

(21) International Application Number: PCT/US2003/025123

(22) International Filing Date: 11 August 2003 (11.08.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/222,307 15 August 2002 (15.08.2002) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 10/222,307 (CIP)
Filed on 15 August 2002 (15.08.2002)

(71) Applicant *(for all designated States except US)*: WASHINGTON UNIVERSITY IN ST. LOUIS [US/US]; One Brooking Drive, St. Louis, MO 63130 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: SCHUEHLER, David, V. [US/US]; 12306 Halsgame Lane, St. Louis, MO 63141 (US). LOCKWOOD, John, W. [US/US]; 839 Jackson Ave., St. Louis, MO 63130 (US).

(74) Agents: RASCHE, Patrick, W. et al.; Armstrong Teasdale LLP, One Metropolitan Square, Sute 2600, St. Louis, MO 63102 (US).

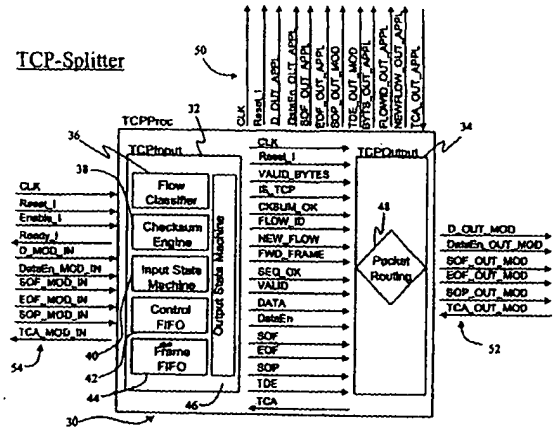
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: TCP-SPLITTER: RELIABLE PACKET MONITORING METHODS FOR HIGH SPEED NETWORKS



(57) **Abstract:** A method for obtaining daata while facilitating keeping a minimum amount of state is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one of an Application Specific Integrated Circuit (ASIC) and a Field Programmable Gate Array FPGA. The method also includes removing an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame with at least one of the ASIC and the FPGA, and determining a validity of a checksum of the removed steam-oriented protocol header. The method also includes dropping the IP frame when the checksum is invalid, supplying a client application with data from the removed protocol frame when the checksum is valid, and sending an IP frame including the removed stream-oriented protocol frame to the third device from the first device when the checksum is valid.

WO 2004/017604 A3

(88) Date of publication of the international search report:
8 July 2004

(15) Information about Correction:
Previous Correction:
see PCT Gazette No. 23/2004 of 3 June 2004, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

National Application No
PCT/US 03/25123

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/26 H04L29/06 H04L12/14

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 6 397 259 B1 (HAWKINS JEFFREY C ET AL) 28 May 2002 (2002-05-28) column 9, line 13 - column 13, line 17 column 18, line 1 - column 20, line 3 column 86, line 53 - column 111, line 67	1,3-9, 11,31,33 2,13-20, 22
A	WO 01/80558 A (SOLIDSTREAMING INC) 25 October 2001 (2001-10-25) column 24, lines 7-13 column 17, line 20 - column 18, line 3 column 6, lines 19-21 column 3, line 10 - column 4, line 2	1-9,11, 13-20, 22,31,33

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
4 May 2004	18. 05. 2004

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Dupuis, H

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 03/25123

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6 397 335 B1 (BRETSCHER JOHN THOMAS ET AL) 28 May 2002 (2002-05-28)	48,49, 51,56, 79,80, 82,87
A	column 3, line 9 - column 5, line 12 column 6, line 30 - column 7, line 65 column 12, lines 15-38 column 13, lines 40-46	
X	WO 02/061525 A (HALL HOWARD ; MERHAR MILAN (US); PIRUS NETWORKS (US); SOKOLINSKI ILIA) 8 August 2002 (2002-08-08) page 4, lines 11-18 page 98, line 1 - page 102, line 32	58,89
A	US 6 412 000 B1 (PACKER ROBERT L ET AL) 25 June 2002 (2002-06-25) column 2, lines 35-43 column 4, lines 5-17 column 12, line 64 - column 13, line 10	58,89

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 03/25123

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this International application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☒ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

1-9,11,13-20,22,31,33,48-51,56,58,79-82,87,89

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-9,11,13-20,22,31,33

   A method and an apparatus, comprising dropping an IP frame when the checksum is invalid, and configured to classify removed protocol frames as at least one of a sequence number greater than expected classification, a sequence number equal to expected classification, a sequence number less than expected classification.

---

2. claims: 10(as dep. on 1),21 (as dep. on 13),35-47,66-78

   Method comprising a client counting bits.

---

3. claims: 12(as dep. on 1),23(as dep. on 13)-30,34,65

   A method, an apparatus, a network, a data transmission system, and an FPGA, comprising determining if a frame contains programming data.

---

4. claim: 32

   Method on a network using a single TCP/IP source and a single destination.

---

5. claims: 48-51,79-82

   Method and apparatus comprising manipulating a stream such that a second device receives data different than that sent from the first device.

---

6. claims: 52-55,57,59-64,83-86,88,90-95

   Method and apparatus comprising using a plurality of content scaning engines.

---

7. claims: 56,87

   Method and apparatus to detect a content match that spans multiple packets.

---

8. claims: 58,89

   Method and apparatus comprising handling overlapping transmissions using a data enabled signal and a valid bytes vector.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

INTERNATIONAL SEARCH REPORT

Information on patent family members

National Application No
PCT/US 03/25123

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6397259 B1	28-05-2002	AU 4210099 A	13-12-1999
		AU 4407899 A	13-12-1999
		CA 2333033 A1	02-12-1999
		CA 2333055 A1	02-12-1999
		EP 1088421 A2	04-04-2001
		EP 1092186 A1	18-04-2001
		GB 2353923 A	07-03-2001
		GB 2357222 A	13-06-2001
		WO 9961984 A1	02-12-1999
		WO 9962268 A2	02-12-1999
WO 0180558 A	25-10-2001	AU 5174801 A	30-10-2001
		WO 0180558 A2	25-10-2001
US 6397335 B1	28-05-2002	US 5987610 A	16-11-1999
		US 2002138766 A1	26-09-2002
		US 2002174350 A1	21-11-2002
		AU 2577399 A	30-08-1999
		WO 9941875 A1	19-08-1999
WO 02061525 A	08-08-2002	AU 2010802 A	15-05-2002
		AU 2719102 A	15-05-2002
		AU 3058502 A	15-05-2002
		AU 4155902 A	18-06-2002
		WO 0237300 A1	10-05-2002
		WO 0246866 A2	13-06-2002
		WO 02061525 A2	08-08-2002
		WO 02069166 A1	06-09-2002
		WO 0237224 A2	10-05-2002
		WO 0237225 A2	10-05-2002
US 6412000 B1	25-06-2002	US 2002143939 A1	03-10-2002
		US 2002055998 A1	09-05-2002
		AU 1421799 A	15-06-1999
		WO 9927684 A1	03-06-1999